MaxSAT Evaluation 2019

Solver and Benchmark Descriptions

Fahiem Bacchus, Matti Järvisalo, and Ruben Martins *(editors)*

University of Helsinki
Department of Computer Science
Series of Publications B
Report B-2019-2

Helsinki 2019

PREFACE

The MaxSAT Evaluations (`https://maxsat-evaluations.github.io`) are a series of events focusing on the evaluation of current state-of-the-art systems for solving optimization problems via the Boolean optimization paradigm of maximum satisfiability (MaxSAT). Organized yearly starting from 2006, the year 2019 brought on the 14th edition of the MaxSAT Evaluations. Some of the central motivations for the MaxSAT Evaluation series are to provide further incentives for further improving the empirical performance of the current state of the art in MaxSAT solving, to promote MaxSAT as a serious alternative approach to solving NP-hard optimization problems from the real world, and to provide the community at large heterogenous benchmark sets for solver development and research purposes. In the spirit of a true evaluation—rather than a competition, unlike e.g. the SAT Competition series—no winners are declared, and *no awards or medals are handed out* to overall best-performing solvers.

The 2019 instantiation of the evaluation series follows closely the revised arrangements brought on by the new organization team in 2017.

The 2019 evaluation consisted of a total of three tracks: two for complete solvers (one for solvers focusing on unweighted and one for solvers focusing on weighted MaxSAT instances) and a special track for incomplete MaxSAT solvers (using two short per-instance time limits, 60 and 300 seconds, differentiating from the per-instance time limit of 1 hour imposed in the main complete tracks). As in 2017-2018, no distinction was made between "industrial" and "crafted" benchmarks, and no track for purely randomly generated MaxSAT instances was organized.

Adhering to the new rules introduced in 2017, solvers were now required to be open-source, and the source codes of all participating solvers were made available online on the evaluation webpages after the evaluation results were presented at the SAT 2019 conference. Furthermore, a 1-2 page solver description was required for each solver submission, to provide some details on the search techniques implemented in the solvers. The solvers descriptions together with descriptions of new benchmarks for 2019 are collected together in this compilation.

Finally, we would like to thank everyone who contributed to MaxSAT Evaluation 2019 by submitting their solvers or new benchmarks. We are also grateful for the computational resources provided by the StarExec initiative which enabled running the 2019 evaluation smoothly.

Fahiem Bacchus, Matti Järvisalo, & Ruben Martins
MaxSAT Evaluation 2019 Organizers

Contents

Benchmark Descriptions

SOLVER DESCRIPTIONS

Complete Solvers

Pacose: An Iterative SAT-based MaxSAT Solver

Tobias Paxian, Sven Reimer, Bernd Becker
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee 051
79110 Freiburg, Germany
{ paxiant | reimer | becker }@informatik.uni-freiburg.de

***Abstract*—Pacose is a SAT-based MaxSAT solver using a CNF encoding for Pseudo-Boolean (PB) constraints [1]. It is an extension of the model guided QMaxSAT1702 [2] solver based on Glucose 3.0 [3] SAT solver. It uses a simple heuristic to choose between the Binary Adder [4] encoding of QMaxSAT and the Dynamic Global Polynomial Watchdog (DGPW) encoding which is based on [5].**

***Index Terms*—MaxSAT Solver, QMaxSAT, Glucose, Dynamic Global Polynomial Watchdog**

I. TITLE

We use a new constraint encoding for PB-constraints solving the weighted MaxSAT problem with iterative SAT-based methods based on the Polynomial Watchdog (PW) CNF encoding called DGPW. The watchdog of the PW encoding indicates whether the bound of the PB constraint holds. In our approach, we lift this static watchdog concept to a dynamic one allowing an incremental convergence to the optimal result. Consequently, we formulate and implement a SAT-based algorithm for our new Dynamic Polynomial Watchdog (DPW) encoding which can be applied for solving the MaxSAT problem. Furthermore, we introduce three fundamental optimizations of the PW encoding also suited for the original version leading to significantly less encoding size.

We integrated this encoding into QMaxSAT (2nd place in the last MaxSAT Evaluation 2017) and adapt the heuristic of QMaxSAT to choose between the Binary Adder encoding of QMaxSAT and our DGPW approach.

REFERENCES

[1] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted maxsat," *Theory and Applications of Satisfiability Testing–SAT 2018*, 2018.
[2] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, pp. 95–100, 2012.
[3] G. Audemard and L. Simon, "On the glucose sat solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.
[4] J. P. Warners, "A linear-time transformation of linear inequalities into conjunctive normal form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.
[5] O. Bailleux, Y. Boufkhad, and O. Roussel, "New encodings of pseudo-boolean constraints into CNF," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2009, pp. 181–194.

This work is partially supported by the DFG project Algebraic Fault Attacks (funding id PO 1220/7-1, BE 1176 20/1, KR 1907/6-1).

QMaxSAT in MaxSAT Evaluation 2018

Aolong Zha
Faculty of Information Science and Electrical Engineering
Kyushu University
744 Motooka, Nishi-ku, Fukuoka, Japan
cyouryuuryuu@gmail.com

QMaxSAT is a satisfiability-based solver, which uses CNF encoding of pseudo-Boolean (PB) constraints [1]. The efficiency of MaxSAT solvers depends on critically on which SAT solver we use and how we encode the PB constraints. The QMaxSAT is obtained by adapting a CDCL based SAT solver Glucose 3.0 [2], [3]. In addition, we introduce a new encoding method, called n-level modulo totalizer encoding in to our solver. This encoding is a hybrid between Modulo Totalizer (MTO) [4] and Weighted Totalizer (WTO) [5], incorporating the idea of mixed radix base [6].

Let $\phi = \{(C_1, w_1), \ldots, (C_m, w_m), C_{m+1}, \ldots, C_{m+m'}\}$ be a MaxSAT [7] instance where C_i is a soft clause with weight w_i $(i = 1, \ldots, m)$ and C_{m+j} is a hard clause $(j = 1, \ldots, m')$. We added a new blocking variable, b_i, to each soft clause $C_i (i = 1, \ldots, m)$. Solving the MaxSAT problem for ϕ is reduced to finding a SAT model of $\phi' = \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$, which minimizes $\sum_{i=1}^{m} w_i b_i$.

Such SAT models are obtained using a SAT solver as follows: Run the SAT solver to get an initial model and calculate $k = \sum_i w_i b_i$ in it, add PB constraint $\sum_i w_i b_i < k$, and run the solver again. If ϕ' is unsatisfiable, then ϕ is also unsatisfiable as the MaxSAT problem. Otherwise, the process is repeated with the new smaller solution. The latest model is a MaxSAT solution of ϕ. QMaxSAT leaves the manipulation of the PB constraints to Glucose by encoding them into SAT.

We introduce a hybrid encoding [8] which inherits modular arithmetic from MTO and distinct combinations of weights from WTO. The latter is essentially the same as Generalized Totalizer, which only generate auxiliary variables for each unique combination of weights. We also enhanced the encoding by multi-level modulo arithmetic based on a mixed radix numeral system [9]. This encoding method always produces a polynomial-size CNF in the number of input variables.

It is important to find a suitable mixed radix base with low time-consumption that reduces the number of auxiliary variables for our new encoding. We select the integer whose rate of divisibility is the highest for all weights[1] as the suitable modulus for each digit. Furthermore, we also add other heuristics tailored in our implementation, such as evaluating and voting for the candidates of modulus, dynamically adjusting the lower limit of the required rate of divisibility, etc.

[1]Before selecting the next modulus, we update all the weights to their quotients of dividing the previous selected modulus.

References

[1] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A Partial Max-SAT Solver," *JSAT*, vol. 8, no. 1/2, pp. 95–100, 2012.

[2] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, C. Boutilier, Ed., 2009, pp. 399–404.

[3] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518.

[4] T. Ogawa, Y. Liu, R. Hasegawa, M. Koshimura, and H. Fujita, "Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*. IEEE Computer Society, 2013, pp. 9–17.

[5] S. Hayata and R. Hasegawa, "Improvement in CNF Encoding of Cardinality Constraints for Weighted Partial MaxSAT," *SIG-FPAI, in Japanese*, vol. 4, no. 04, pp. 85–90, 2015.

[6] M. Codish, Y. Fekete, C. Fuhs, and P. Schneider-Kamp, "Optimal Base Encodings for Pseudo-Boolean Constraints," in *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, ser. Lecture Notes in Computer Science, P. A. Abdulla and K. R. M. Leino, Eds., vol. 6605. Springer, 2011, pp. 189–204.

[7] C. M. Li and F. Manyà, "MaxSAT, Hard and Soft Constraints," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 185, pp. 613–631.

[8] A. Zha, M. Koshimura, and H. Fujita, "A Hybrid Encoding of Pseudo-Boolean Constraints into CNF," in *Conference on Technologies and Applications of Artificial Intelligence, TAAI 2017, Taipei, Taiwan, December 1-3, 2017*. IEEE, 2017, pp. 9–12.

[9] A. Zha, N. Uemura, M. Koshimura, and H. Fujita, "Mixed Radix Weight Totalizer Encoding for Pseudo-Boolean Constraints," in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence, Boston, MA, USA, November 6-8, 2017*. IEEE Computer Society, 2017, pp. 868–875.

UWrMaxSat - a new MiniSat+-based Solver in MaxSAT Evaluation 2019

Marek Piotrów
Institute of Computer Science, University of Wrocław
Wrocław, Poland
marek.piotrow@uwr.edu.pl

***Abstract*—UWrMaxSat is a new MiniSat+-based solver participating in MaxSAT Evaluation 2019. It has been created recently at the University of Wrocław. It is a complete solver for partial weighted MaxSAT instances. It incrementally uses COMiniSatPS by Chanseok Oh (2016) as an underlying SAT solver, but may be compiled with other MiniSat-like solvers. It was developed on the top of our PB-solver (called kp-minisatp) that was presented at Pragmatics of SAT 2018 and which is an extension of the well-known MiniSat+ solver. In its main configuration, UWrMaxSat applies an unsatisfiability-core-based OLL procedure and uses the kp-minisatp sorter-based pseudo-Boolean constraint encoding to translate new cardinality constraints into CNF.**

***Index Terms*—MaxSAT-solver, UWrMaxSAT, COMiniSatPS, sorter-based encoding, core-guided, complete solver**

I. Introduction

At Pragmatics of SAT 2018 workshop, Michał Karpiński and Marek Piotrów presented a new pseudo-Boolean constraint solver called kp-minisatp [8] that was created as an extension of MiniSat+ 1.1 solver by Eén and Sörensson (2012) [6]. In the solver we replaced the encoding based on odd-even sorting networks by a new one using our construction of selection networks called 4-Way Merge Selection Networks [9]. We also optimized mixed radix base searching procedure and added a few other optimizations based on literature. Our experiments showed that the solver is competitive to other state-of-art solvers.

Believing that the encoding can be also used in MaxSAT solvers, I have implemented such a solver on the top of kp-minisatp and the result called UWrMaxSat is submitted to MaxSAT Evaluation 2019.

II. Description

The solver is prepared to be compiled with one of a few MiniSat-like SAT solvers: COMiniSatPS by Chanseok Oh (2016), Glucose 4.1 (2016) and 3.0 (2013) by Gilles Audemard and Laurent Simon [3], and original Minisat 2.2 (2010) by Niklas Eén and Niklas Sörensson [4]. The first one was selected as the default SAT solver and it is used incrementally with the help of assumptions [5]. Other MiniSat-like SAT solvers can be also applied, because the interface between UWrMaxSAT and a SAT solver is small and defined by MiniSat.

Three different core-guided searching strategies have been implemented in the solver: a linear unsat-sat one, a linear sat-unsat one and a binary search sat-unsat one. The first one was selected as default and it was optimized much more than the others. In order to process unsatisfiability cores in the default strategy we select the OLL procedure [1] and encode its cardinality constraints by our encoding based on 4-Way Merge Selection Network. The encoding uses also Direct Networks for constraints with a small number of literals [2]. A core is minimized [11] before it is converted into a cardinality constraint.

In case of weighted instances, several other optimization techniques are applied:

- A stratification technique: Soft clauses are sorted and grouped by weights; the groups of relaxed soft clauses are delivered to the SAT solver gradually, starting from ones with the largest weights. We use only one relaxation variable per non-unit soft clause.
- A hardening technique: The algorithm refines both lower and upper bounds on the weight of a solution. Base on their values, the heaviest soft clauses can be transformed into hard ones.
- A BMO technique [10]: Some weighted instances can encode multi-objective problems and the optimality criterion is lexicographic. Such instances are detected and the search procedure is optimized accordingly.
- A preprocessing technique: Soft clauses can be preprocess to detect unit cores and at-most-one cores. Such cores are encoded in a more efficient way (implemented based on the code of RC2, see [7]).
- A mixed strategy technique: If the linear unsat-sat searching is unsuccessful for a predefined time, it can be switched to the binary searching without restarting the SAT solver.

Finally, the solver can deal with unbounded integer weights when it is compiled with the -D BIG_WEIGHTS option.

III. Acknowledgments

I would like to thank Chankseok Oh for allowing me to use COMiniSatPS in the MaxSAT Evaluation. I would like also to thank Niklas Eén and Niklas Sörensson for the development of MiniSat 2.2 and MiniSat+ 1.1.

References

[1] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12*, volume 17, pages 212–221, 2012.

[2] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[3] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[4] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[5] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543 – 560, 2003. BMC'2003, First International Workshop on Bounded Model Checking.

[6] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[7] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

[8] Michał Karpiński and Marek Piotrów. Competitive sorter-based encoding of pb-constraints into sat. In Daniel Le Berre and Matti Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 65–78. EasyChair, 2019.

[9] Michał Karpiński and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints*, Apr 2019.

[10] Joao Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Annals of Mathematics and Artificial Intelligence*, 62(3):317–343, Jul 2011.

[11] João P. Marques Silva. Minimal unsatisfiability: Models, algorithms and applications (invited paper). pages 9–14, 01 2010.

MaxHS in the 2019 MaxSat Evaluation

Fahiem Bacchus
Department of Computer Science
University of Toronto
Ontario, Canada
Email: fbacchus@cs.toronto.edu

1. MaxHS

MaxHS is a MaxSat solver that originated in the PhD work of Davies [4]. It was the first MaxSat solver to utilize the Implicit Hitting Set (IHS) approach, and its core components are described in [4], [2], [3], [5]. Additional useful insights into IHS are provided in [7], [8]. IHS solvers utilize both an integer programming (IP) solver and a SAT solver in a hybrid approach to MaxSat solving. MaxHS utilizes minisat v2.2 as its SAT solver and IBM's CPLEX v12.8 as its IP solver. Interestingly experiments with more sophisticated SAT solvers like Glucose http://www.labri.fr/perso/lsimon/glucose/ and Lingeling http://fmv.jku.at/lingeling/ yielded inferior performance. This indicates that the SAT problems being solved are quite simple, too simple for the more sophisticated techniques used in these SAT solvers to pay off. In fact, simpler SAT problems are one of the original motivations behind MaxHS [2].

The main change in the 2019 version of MaxHS were changes made to the Minisat assumption mechanism. In particular, the modifications described in [6] were implemented.

1.0.1. Termination based on Bounding. MaxHS maintains an upper bound (and best model found so far) and a lower bound on the cost of an optimal solution (the IP solver computes valid lower bounds). MaxHS terminates when the gap between the lower bound and upper bound is low enough (with integer weights when this gap is less than 1, the upper bound model is optimal). This means that MaxHS no longer needs to wait until the IP solver returns an hitting set whose removal from the set of soft clauses yields SAT; it can return when the IP solver's best lower bound is close enough to show that the best model is optimal.

1.0.2. Early Termination of Cplex. In previous versions of MaxHS, the IP solver was run to completion forcing it to find an optimal solution every time it is called. However, with bounding, optimal solutions are not always needed. In particular, if the IP solver finds a feasible solution whose cost is better than the current best model it can return that: either the IP solution is feasible for the MaxSat problem, in which case we can lower the upper bound, or it is infeasible in which case we can obtain additional cores to augment the IP model (and thus increase the lower bound). Terminating the IP solver before optimization is complete can yield significant time savings.

1.0.3. Reduced Cost fixing via the LP-Relaxation. Using an LP relaxation and the reduced costs associated with the optimal LP solution, some soft clauses can be hardened or immediately falsified. See [1] for more details.

1.0.4. Mutually Exclusive Soft Clauses. Sets of soft clauses of which at most one can be falsified or at most one can be satisfied are detected. When all of these soft clauses have the same weight they can all be more compactly encoded with a single soft clause. This encoding does not always yield better performance due to some subtle effects. However, techniques were developed to better exploit such information, and a fuller description of these techniques is in preparation. With these techniques performance gains were achieved.

1.0.5. Other clauses to the IP Solver. Problems with a small number of variables are given entirely to the IP solver, so that it directly solves the MaxSat problem. In this case the SAT solver is used to first compute some additional clauses and cores, and to find a better initial model for the IP solver. This additional information from the SAT solver often makes the IP solver much faster than just running the IP solver and represents an alternate way of hybridizing SAT and IP solvers.

1.0.6. Other techniques for finding Cores. MaxHS iteratively calls the IP solver to obtain a hitting set of the cores computed so far. If that hitting set does not yield an optimal MaxSat solution then more cores must be added to the IP solver. In some of these iterations very few cores can be found causing only a slight improvement to the IP solver's model. This results in a large number of time consuming calls to the IP solver. Two method were developed to aid this situation (a) we ask the IP solver for more solutions and generate cores from these as hitting sets as well and (b) if we have a new upper bound model we try to improve this model by converting it to a minimal correction set (MCS). In converting the upper bound model to an MCS we either find a better model (lowering the upper bound) or we compute additional conflicts that can be added to the IP solver.

References

[1] Bacchus, F., Hyttinen, A., Järvisalo, M., Saikko, P.: Reduced cost fixing in maxsat. In: Proc. CP. p. in press (2017)

[2] Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Proc. CP. Lecture Notes in Computer Science, vol. 6876, pp. 225–239. Springer (2011)

[3] Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MaxSAT. In: Proc. SAT. Lecture Notes in Computer Science, vol. 7962, pp. 166–181. Springer (2013)

[4] Davies, J.: Solving MAXSAT by Decoupling Optimization and Satisfaction. Ph.D. thesis, University of Toronto (2013), http://www.cs.toronto.edu/~jdavies/Davies_Jessica_E_201311_PhD_thesis.pdf

[5] Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Proc. CP. Lecture Notes in Computer Science, vol. 8124, pp. 247–262. Springer (2013)

[6] Hickey, R., Bacchus, F.: Speeding up assumption-based sat. In: Proceedings of Theory and Applications of Satisfiability Testing - SAT 2019, 22nd International Conference, SAT 2009, Lisbon, Portugal (2019)

[7] Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid MaxSAT solver. In: Proc. SAT. Lecture Notes in Computer Science, vol. 9710, pp. 539–546. Springer (2016)

[8] Saikko, P.: Re-implementing and Extending a Hybrid SAT-IP Approach to Maximum Satisfiability. Master's thesis, University of Helsinki (2015), http://hdl.handle.net/10138/159186

Maxino

Mario Alviano
Department of Mathematics and Computer Science
University of Calabria
87036 Rende (CS), Italy
Email: alviano@mat.unical.it

***Abstract*—Maxino is based on the k-ProcessCore algorithm, a parametric algorithm generalizing OLL, ONE and PMRES. Parameter k is dynamically determined for each processed unsatisfiable core by a function taking into account the size of the core. Roughly, k is in $O(\log n)$, where n is the size of the core. Satisfiability of propositional theories is checked by means of a pseudo-boolean solver extending Glucose 4.1 (single thread).**

A Very Short Description of the Solver

The solver MAXINO is build on top of the SAT solver GLUCOSE [7] (version 4.1). MaxSAT instances are normalized by replacing non-unary soft clauses with fresh variables, a process known as *relaxation*. Specifically, the relaxation of a soft clause ϕ is the clause $\phi \vee \neg x$, where x is a variable not occurring elsewhere; moreover, the weight associated with clause ϕ is associated with the soft literal x. Hence, the normalized input processed by MAXINO comprises hard clauses and soft literals, so that the computational problem amounts to maximize a linear function, which is defined by the soft literals, subject to a set of constraints, which is the set of hard clauses.

The algorithm implemented by MAXINO to address such a computational problem is based on unsatisfiable core analysis, and in particular takes advantage of the following *invariant*: A model of the constraints that satisfies all soft literals is an optimum model. The algorithm then starts by searching such a model. On the other hand, if an inconsistency arises, the unsatisfiable core returned by the SAT solver is analyzed. The analysis of an unsatisfiable core results into new constraints and new soft literals, which replace the soft literals involved in the unsatisfiable core. The new constraints are essentially such that models satisfying all new soft literals actually satisfy all but one of the replaced soft literals. Since there is no model that satisfies all replaced soft literals, it turns out that the invariant is preserved, and the process can be iterated.

Specifically, the algorithm implemented by MAXINO is K, based on the k-ProcessCore procedure introduced by Alviano et al. [2]. It is a parametric algorithm generalizing OLL [3], ONE [2] and PMRES [8]. Intuitively, for an unsatisfiable core $\{x_0, x_1, x_2, x_3\}$, ONE introduces the following constraint:

$$x_0 + x_1 + x_2 + x_3 + \neg y_1 + \neg y_2 + \neg y_3 \geq 3$$
$$y_1 \rightarrow y_2 \quad y_2 \rightarrow y_3$$

where y_1, y_2, y_3 are fresh variables (the new soft literals that replace x_0, x_1, x_2, x_3). OLL introduces the following constraints (the first immediately, the second if a core containing y_1 is subsequently found, and the third if a core containing y_2 is subsequently found):

$$x_0 + x_1 + x_2 + x_3 + \neg y_1 \geq 3$$
$$x_0 + x_1 + x_2 + x_3 + \neg y_2 \geq 2$$
$$x_0 + x_1 + x_2 + x_3 + \neg y_3 \geq 1$$

Concerning PMRES, it introduces the following constraints:

$$x_0 \vee x_1 \vee \neg y_1 \quad z_1 \leftrightarrow x_0 \wedge x_1$$
$$z_1 \vee x_2 \vee \neg y_2 \quad z_2 \leftrightarrow z_1 \wedge x_2$$
$$z_2 \vee x_3 \vee \neg y_3$$

which are essentially equivalent to the following constraints:

$$x_0 + x_1 + \neg z_1 + \neg y_1 \geq 2 \quad z_1 \rightarrow y_1$$
$$z_1 + x_2 + \neg z_2 + \neg y_2 \geq 2 \quad z_2 \rightarrow y_2$$
$$z_2 + x_3 \quad + \neg y_3 \geq 1$$

where y_1, y_2, y_3 are fresh variables (the new soft literals that replace x_0, x_1, x_2, x_3), and z_1, z_2 are fresh auxiliary variables.

Algorithm K, instead, introduces a set of constraints of bounded size, where the bound is given by the chosen parameter k, and is specifically $2 \cdot (k+1)$. ONE, which is essentially a smart encoding of OLL, is the special case for $k = \infty$, and PMRES is the special case for $k = 1$. For the example unsatisfiable core, another possibility is $k = 2$, which would results in the following constraints:

$$x_0 + x_1 + x_2 + \neg z_1 + \neg y_1 + \neg y_2 \geq 3 \quad z_1 \rightarrow y_1 \quad y_1 \rightarrow y_2$$
$$z_1 + x_3 \quad + \neg y_3 \quad \geq 1$$

In this version of MAXINO, the parameter k is dynamically determined based on the size of the analyzed unsatisfiable core: $k \in O(\log n)$, where n is the size of the core.

The analysis of unsatisfiable core is preceded by a *shrink* procedure [1]. Specifically, a reiterated progression search is performed on the unsatisfiable core returned by the SAT solver. Such a procedure significantly reduces the size of the unsatisfiable core, even if it does not necessarily returns an unsatisfiable core of minimal size. Additionally, satisfiability checks performed during the shrinking process are subject to a budget on the number of conflicts, so that the overhead due to hard checks is limited. Specifically, the budget is set to the number of conflicts arose in the satisfiability check that lead to detecting the unsatisfiable core; if such a number is less than 1000 (one thousand), the budget is raised to 1000. The budget is divided by 2 every time the progression is reiterated.

Weighted instances are handled by *stratification* and introducing *remainders* [4]–[6]. Specifically, soft literals are

partitioned in strata depending on the associated weight. Initially, only soft literals of greatest weight are considered, and soft literals in the next stratum are added only after a model satisfying all considered soft literals is found. When an unsatisfiable core is found, the weight of all soft literals in the core is decreased by the weight associated with last added stratum. Soft literals whose weight become zero are not considered soft literals anymore.

Finally, a preprocessing step is performed on unweighted instances, which essentially iterates on all hard clauses of the input theory, sorted by length, and checks whether they already witness some unsatisfiable core. Specifically, an hard clause witnesses an unsatisfiable core if all literals in the clause are the complement of a soft literal. If this is the case, the unsatisfiable core is analyzed immediately. The rationale for such a preprocessing step is that hard clauses in the input theory are often small, and the smaller the better for the unsatisfiable core based algorithms.

REFERENCES

[1] Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *TPLP*, 16(5-6):533–551, 2016.

[2] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A maxsat algorithm using cardinality constraints of bounded size. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2677–2683. AAAI Press, 2015.

[3] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *28th International Conference on Logic Programming*, pages 211–221, Budapest, Hungary, September 2012.

[4] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *SAT 2009*, pages 427–440, Swansea, UK, June 2009. Springer.

[5] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196(0):77–105, March 2013.

[6] Josep Argelich, Inês Lynce, and João P. Marques Silva. On solving boolean multilevel optimization problems. In *21st International Joint Conference on Artificial Intelligence*, pages 393–398, Pasadena, California, July 2009. IJCAI Organization.

[7] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *21st International Joint Conference on Artificial Intelligence*, pages 399–404, Pasadena, California, July 2009. IJCAI Organization.

[8] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2717–2723, Québec City, Canada, July 2014. AAAI Press.

Open-WBO @ MaxSAT Evaluation 2019

Ruben Martins
rubenm@cs.cmu.edu
CMU, USA

Norbert Manthey
nmanthey@conp-solutions.com
Dresden, Germany

Miguel Terra-Neves, Vasco Manquinho, Inês Lynce
{neves,vmm,ines}@inesc-id.pt
INESC-ID/IST, Portugal

I. Introduction

Open-WBO [1] is an open source MaxSAT solver that supports several MaxSAT algorithms [2], [3], [4], [5], [6], [7], [8] and SAT solvers [9], [10], [11]. Open-WBO is particularly efficient for unweighted MaxSAT and has been one of the best solvers in the MaxSAT Evaluations from 2014 to 2017. Three versions of Open-WBO were submitted to the MaxSAT Evaluation 2019: open-wbo-g, open-wbo-ms and open-wbo-ms-pre. The remainder of this document describes the differences between these versions.

II. SAT solvers

Open-WBO is based on the data structures of MiniSat 2.2 [9], [12]. Therefore, solvers based on MiniSat 2.2 can be used as a potential back-end solvers. For the MaxSAT Evaluation 2019, we use Glucose 4.1 [10], [13], [14] as the back-end SAT solver of the open-wbo-g version and MergeSAT [11] as the back-end SAT solver of the versions open-wbo-ms and open-wbo-ms-pre.

MergeSAT is a new CDCL solver developed by Norbert Manthey and it is based on the SAT competition winner of 2018, MapleLCMDistChronoBT [15], and adds several known techniques. For restarts, only partial backtracking is used, learned clause minimization is implemented more efficiently, and also applies simplification again in case the first swipe resulted in a simplification. Finally, the time-based decision heuristic switch is made deterministic by using solving steps. To support being used inside MaxSAT solvers, the incremental search feature had to been enabled again.

III. MaxSAT Algorithms

In this section we briefly describe the algorithms used for the complete and incomplete tracks at the MSE2019.

A. Complete Track

For the complete track, Open-WBO uses a variant of the unsatisfiability-based algorithm MSU3 [3] and the OLL algorithm [7]. These algorithms work by iteratively refining a lower bound λ on the number of unsatisfied soft clauses until an optimum solution is found. Both MSU3 and OLL use the Totalizer encoding for incremental MaxSAT solving [4]. For unweighted MaxSAT, we extended the incremental MSU3 algorithm [4] with resolution-based partitioning techniques [8]. We represent a MaxSAT formula using a resolution-based graph representation and iteratively join partitions by using a proximity measure extracted from the graph representation of the formula. The algorithm ends when only one partition remains and the optimal solution is found. Since the partitioning of some MaxSAT formulas may be unfeasible or not significant, we heuristically choose to run either MSU3 with partitions or the OLL algorithm. In particular, we do not use partition-based techniques when one of the following criteria is met: (i) the formula is too large ($>$ 1,000,000 clauses), (ii) the ratio between the number of partitions and soft clauses is too high (> 0.8), (iii) the sparsity of the graph is too small (< 0.04), or (iv) there exist some at-most-one relations between soft clauses (> 10), i.e. if one soft clause is satisfied it implies that some other soft clauses will be unsatisfied.

For weighted MaxSAT, we use a variant of the OLL algorithm [7] without optimizations. Potential avenues for improvements involve reusing the soft cardinality via assumptions instead of cloning them [16], extending the OLL algorithm to use lexicographic optimization [17], and perform core minimization.

B. Incomplete Track

For the unweighted incomplete track, Open-WBO uses an incremental variant of the MSU4 algorithm [18], [19] with the incremental Totalizer encoding [4]. This is a complete MaxSAT algorithm that performs a linear search SAT-UNSAT but lazily expands the soft clauses that can be relaxed, i.e. unsatisfied. This approach is particularly effective for benchmarks with thousands of soft clauses [19].

For the weighted complete track, Open-WBO uses a variant of the lexicographical optimization algorithm [17] that does not guarantee optimality [20], [21]. This algorithm considers n objective functions where n is the number of distinct weights in the MaxSAT formula. This is done by performing a sequence of calls to a SAT solver and refining an upper bound μ on the number of unsatisfied soft clauses. To restrict μ at each iteration, we need to encode cardinality constraints into CNF, for which, incremental Totalizer encoding [4] has been used. Once for a given objective function the upper bound μ cannot be improved, it is frozen, and the next objective function in the order is optimized. If an optimal solution is found when using this algorithm, then it is not necessarily an optimal solution of the input formula. Once this happens, we change to a complete algorithm based on linear search SAT-UNSAT that uses the Adder [22] or Generalized Totalizer encoding (GTE) [23] to encode pseudo-Boolean constraints. In order to maintain the lexicographic structure as long as possible, we

only relax the previous lexicographical restrictions if they are the reason for unsatisfiability.

IV. Preprocessing

We integrated the MaxSAT preprocessor MaxPre [24] with Open-WBO via MaxPre API into the version open-wbo-ms-pre. To avoid spending too much time in preprocessing, we limit the number of tries for each preprocessing technique with the flag `-skiptechnique=100` and the time limit taken by the preprocessor to 10% of the total time (or 180 seconds if smaller). MaxPre can simplify the formula using a variety of techniques, such as, blocked clause elimination, unit propagation, bounded variable elimination, subsumption elimination, self subsuming resolution, subsumed label elimination, binary core removal, bounded variable addition, group subsumed label elimination, equivalence elimination, unhiding techniques, structure labeling and failed label probing.

In addition to the simplifications performed by MaxPre, we also perform identification of unit cores and at-most-one relations between soft clauses by using unit propagation. A similar technique is done in RC2 [25], the winner of the MaxSAT Evaluation 2018.

V. Availability

The latest release of Open-WBO is available under a MIT license in GitHub at *https://github.com/sat-group/open-wbo*.

Acknowledgments

We would like to thank Laurent Simon and Gilles Audemard for allowing us to use Glucose 4.1 in the MaxSAT Evaluation. We would also like to thank Niklas Eén and Niklas Sörensson for the development of MiniSat 2.2. We would like to thank Jeremias Berg and Matti Järvisalo for their help with the MaxSAT preprocessor MaxPre. Additionally, we would like to thank all the collaborators on previous versions of Open-WBO, namely Saurabh Joshi and Mikoláš Janota. Finally, we would like to thank David Chen for his study on the impact of disjoint cores, unit cores, and at-most-one relations between soft clauses that were done in the scope of Independent Studies at CMU.

References

[1] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.

[2] V. Manquinho, J. Marques-Silva, and J. Planes, "Algorithms for Weighted Boolean Optimization," in *SAT*. Springer, 2009, pp. 495–508.

[3] J. Marques-Silva and J. Planes, "On Using Unsatisfiability for Solving Maximum Satisfiability," *CoRR*, 2007.

[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.

[5] R. Martins, V. Manquinho, and I. Lynce, "On Partitioning for Maximum Satisfiability," in *ECAI*. IOS Press, 2012, pp. 913–914.

[6] R. Martins, V. M. Manquinho, and I. Lynce, "Community-based partitioning for maxsat solving," in *SAT*. Springer, 2013, pp. 182–191.

[7] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-Guided MaxSAT with Soft Cardinality Constraints," in *CP*. Springer, 2014, pp. 564–573.

[8] M. Neves, R. Martins, M. Janota, I. Lynce, and V. M. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.

[9] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.

[10] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.

[11] N. Manthey, "Mergesat," in *Proceedings of SAT Competition 2019: Solver and Benchmark Descriptions*, 2019.

[12] N. Sörensson, N. Een, and N. Manthey. (2018, May) GitHub repository for MiniSat. https://github.com/conp-solutions/minisat.

[13] G. Audemard, J.-M. Lagniez, and L. Simon, "Improving glucose for incremental sat solving with assumptions: Application to mus extraction," in *SAT*. Springer, 2013.

[14] G. Audemard and L. Simon. (2018, May) Glucose's home page. http://www.labri.fr/perso/lsimon/glucose.

[15] A. Nadel and V. Ryvchin, "Chronological backtracking," in *SAT*. Springer, 2018, pp. 111–121.

[16] J. Berg and M. Järvisalo, "Weight-Aware Core Extraction in SAT-Based MaxSAT Solving," in *CP*. Springer, 2017, pp. 652–670.

[17] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.

[18] J. Marques-Silva and J. Planes, "Algorithms for maximum satisfiability using unsatisfiable cores," in *DATE*. ACM, 2008, pp. 408–413.

[19] R. Martins, V. Manquinho, and I. Lynce, "Improving linear search algorithms with model-based approaches for maxsat solving," *J. Exp. Theor. Artif. Intell.*, vol. 27, no. 5, pp. 673–701, 2015. [Online]. Available: https://doi.org/10.1080/0952813X.2014.993508

[20] S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation Strategies for Incomplete MaxSAT," in *CP*. Springer, 2018.

[21] S. Joshi, P. Kumar, S. Rao, and R. Martins, "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT," in *JSAT*. IOS Press, 2019.

[22] J. P. Warners, "A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form," *Information Processing Letters*, vol. 68, no. 2, pp. 63–69, 1998.

[23] S. Joshi, R. Martins, and V. M. Manquinho, "Generalized Totalizer Encoding for Pseudo-Boolean Constraints," in *CP*. Springer, 2015, pp. 200–209.

[24] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo, "MaxPre: An Extended MaxSAT Preprocessor," in *SAT*. Springer, 2017, pp. 449–456.

[25] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python Toolkit for Prototyping with SAT Oracles," in *Proc. SAT*, ser. Lecture Notes in Computer Science, O. Beyersdorff and C. M. Wintersteiger, Eds., vol. 10929. Springer, 2018, pp. 428–437.

RC2: a Python-based MaxSAT Solver

Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva
Faculty of Sciences, University of Lisbon, Portugal
{aignatiev,ajmorgado,jpms}@ciencias.ulisboa.pt

I. Introduction

RC2 is an open-source MaxSAT solver written in Python and based on the PySAT framework[1] [1]. It is designed to serve as a simple example of how SAT-based problem solving algorithms can be implemented using PySAT while sacrificing just a little in terms of performance. In this sense, RC2 can be seen as a solver prototype and can be made somewhat more efficient if implemented in a low-level language. RC2 is written from scratch and implements the OLLITI (or *RC2*, i.e. *relaxable cardinality constraints*) MaxSAT algorithm [2], [3] originally implemented in the MSCG MaxSAT solver [3], [4]. The RC2 algorithm proved itself efficient in the previous editions of the MaxSAT Evaluation: namely in 2014, 2015, and 2016 (see the results of the MSCG solver, which was one of the best complete MaxSAT solvers in the aforementioned competitions).

II. Description

RC2 supports *incrementally* a variety of SAT solvers provided by PySAT, and its competition version uses Glucose 3.0 [5] as an underlying SAT oracle. Two variants of the solver were submitted to the MaxSAT Evaluation 2018 including RC2-A and RC2-B. Both of these versions implement the same algorithm [2], [3] and share most of the techniques used [3]. Their major components and differences are briefly described below.

III. Variants of the Solver

The following heuristics are used by both solver variants submitted to the MaxSAT Evaluation 2018: incremental SAT solving [6], Boolean lexicographic optimization [7] and stratification [8] for weighted instances, unsatisfiable core exhaustion (originally referred to as cover optimization) [8].

Additionally, the following heuristic was used in both variants of RC2: given a set S of soft clauses, a number of subsets $S' \subseteq S$ were identified such that at most one soft clause in S' can be satisfied, i.e. $\sum_{c \in S'} c \leq 1$. Every subset S' can be treated as an unsatisfiable core of cost $|S'| - 1$, which can be represented as a single clause.

The only difference between the solver variants is the policy for unsatisfiable core minimization. In contrast to RC2-A, RC2-B applies heuristic unsatisfiable core minimization done with a simple deletion-based *minimal unsatisfiable subset* (*MUS*) extraction algorithm [9]. During the core minimization phase in RC2-B, all SAT calls are dropped after obtaining 1000 conflicts. Note that core minimization in RC2-B is disabled for large *plain* MaxSAT formulas, i.e. those having no hard clauses but more than 100000 soft clauses. The reason is that having this many soft clauses (and, thus, as many assumption literals) and no hard clauses is deemed to make SAT calls too expensive. Although core minimization is disabled in RC2-A, reducing the size of unsatisfiable cores can be still helpful for weighted instances due to the nature of the OLLITI/RC2 algorithm, i.e. because of the clause splitting applied to the clauses of an unsatisfiable core depending on their weight. Therefore, when dealing with weighted instances RC2-A *trims* unsatisfiable cores at most 5 times (e.g. see [3] for details) aiming at getting rid of unnecessary clauses. Note that core trimming is disabled in RC2-A for unweighted MaxSAT instances and it is not used in RC2-B at all.

[1] http://pysathq.github.io

IV. Availability

RC2 is distributed as a part of the PySAT framework, which is available under an MIT license at https://github.com/pysathq/pysat. It can also be installed as a Python package from PyPI:

```
pip install python-sat
```

The RC2 solver can be used as a standalone executable `rc2.py` and can also integrated into a complex Python-based problem solving tool, e.g. using the standard *import* interface of Python:

```
from pysat.examples import rc2
```

References

[1] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: a Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, to appear.
[2] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *CP*, 2014, pp. 564–573.
[3] A. Morgado, A. Ignatiev, and J. Marques-Silva, "MSCG: Robust core-guided MaxSAT solving," *JSAT*, vol. 9, pp. 129–134, 2015.
[4] A. Ignatiev, A. Morgado, V. M. Manquinho, I. Lynce, and J. Marques-Silva, "Progression in maximum satisfiability," in *ECAI*, 2014, pp. 453–458.
[5] G. Audemard, J. Lagniez, and L. Simon, "Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *SAT*, 2013, pp. 309–317.
[6] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electr. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.
[7] J. Marques-Silva, J. Argelich, A. Graca, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence (AMAI)*, vol. 62, no. 3-4, pp. 317–343, 2011.
[8] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving WPM2 for (weighted) partial maxsat," in *CP*, 2013, pp. 117–132.
[9] J. M. Silva, "Minimal unsatisfiability: Models, algorithms and applications (invited paper)," in *ISMVL*, 2010, pp. 9–14.

Incomplete Solvers

LinSBPS

Emir Demirović and Peter J. Stuckey
Department of Computing and Information Systems
University of Melbourne
Australia
(emir.demirovic ∨ pstuckey) @ unimelb.edu.au

I. Introduction

The solver was created with the intention to study the effectiveness of local search inspired techniques for maxSAT. This is a long-term goal where the aim is to develop algorithms that use techniques similar to those of metaheuristics, in particular large neighbourhood search, but within a complete algorithmic setting. Thus, the overall objective would be to improve the *anytime* performance of solvers, which is especially important for large-scale problems where optimality guarantees seem impractical.

II. The Algorithm

We start with the linear MaxSAT algorithm [1]. It computes the optimal solution to a maxSAT problem by repeatedly solving a series of SAT problems, each time adding constraints that force the new solution to be better than than the previously computed one. This algorithm, implemented in Open-WBO [2] with Glucose [3] as the backend solver, was the best solver for the *60 seconds unweighted incomplete track* in the last maxSAT evaluation 2017. However, it was outperformed in the same category with 300 seconds and did not provide competitive solutions for many benchmarks in the weighted incomplete track. The internal SAT solveris a complete backtracking algorithm: it selects a variable, assigns it a truth value, and then either backtracks if a conflict is found or recursively repeats the procedure.

Our approach uses the linear MaxSAT algorithm augmented with two important components: solution-based phase saving and varying resolution technique, where we start considering the problem in *low resolution* and with time increase the resolution.

A. Solution-Based Phase Saving

The variable selection process partially mimics strategies used in local search algorithms: it selects a variable that was frequently involved in recent conflicts (*high activity*, the VSIDS scheme [4]). However, the truth value assignment procedure does not: it is based on *phase saving*, meaning it assigns the value used most recently for the variable. While phase saving is effective for pure SAT problems, *solution-based phase saving* has proven to be more efficient for optimisation [5], where the assignment is based on the best solution found so far. If the previous search was in a space where no better solution exists, time is effectively wasted with standard phase saving. Solution phase saving avoids this by searching around the best solution found. This is reminiscent of local search, as the algorithm is directed *near* the best solution. It can also be seen as a kind of Large Neighbourhood Search [6]. Indeed, assigning values to a set of variables based on the current best soluton and optimising for the remaining variables is a common strategy in metaheuristic algorithms and has been used for decades. Such a technique is particularly relevant for the incomplete track in the maxSAT competition, where solvers are expected to deliver high quality solution within tight time budgets.

To boost its performance, we incorporated solution-based phase saving in the linear algorithm. Solution-based phase saving is not widely used in MaxSAT solving. It is used by WPM3 [7] in a core-guided approach. However, we argue that the technique is more natural for a linear algorithm. As noted, the basic idea has been used in metaheuristic algorithms and even in MaxSAT solving [5] [7], but the position of the linear algorithm with solution-based phase saving among modern MaxSAT solvers is not clear. Thus, we implemented solution-based phase saving in Open-WBO [2] and evaluated its performance using benchmarks from the recent maxSAT evaluation 2017 and the international timetabling competition 2011. We do not present the results of our study in this short paper, but we do note that it provided an improvement over the baseline linear algorithm. In our recent CP paper [8], we studied solution-based phase saving for constraint programming solvers and its relation to *automated large neighbourhood search*. For CP, it provides substantial improvements. We note that we have investigated other phase saving variants, but as of now, the results remain inconclusive.

B. Varying Resolution Approach

While solution-based phase saving does provide improvements, especially for certain classes of problems, it cannot be used effectively for a large set of the MaxSAT competition benchmarks. The reason is that the linear algorithm relies on encoding a single large cardinality constraint, which is directly dependant of the magnitude of the sum of the weights of soft clauses. As the sum grows, in the general case, so does the number of clauses and auxiliary variables that are needed to encode the cardinality constraint. Thus, the memory requirements can be significant. This has a direct impact on the performance of the linear algorithm and it some cases it completely dominates the solver. We note that this is not necessarily the case for core-guided approaches, which for a

large part are uneffected by the magnitude of the weights. Hence, we developed a simplification strategy where we initially consider the problem in *low resolution* where the weights of the MaxSAT problem are divided by a large value. After the simplified problem is solved optimally, the resolution of the problem is increased i.e. the division value is lowered. This continues iteratively until the full original problem is solved. Therefore, the technique is theoretically complete, but in practice for the benchmarks from the last MaxSAT evaluation and the short time limits, only one or two resolutions are typically considered. We note that solution-based phase saving is used during the algorithm, as well as in between resolutions. With this technique, intuitively, the most important constraints are dealt with in the beginning and with executation time other increasing important constraints are added the clause database, resembling local search style methods. It is related to the lexicographical optimisation approach for MaxSAT [9].

The main advantage is that the cardinality constraint that needs to be encoded is orders of magnitude smaller than from the original problem, offering substantial speed-ups. However, the varying resolution approach comes at the price of precision, as an optimum solution for the low-resolution problem does not necessarily correspond to the optimum for the higher resolutions and vice versa. Moreover, given two models for the low-resolution problem and their cost, it is not possible to determine which one of them is better based on their cost without consider the complete original problem. The tendency, heuristically speaking, is that better solutions to the low-resolution problem correlate with better solutions to the original problem.

III. Conclusion

We presented LinSBPS, the algorithm we submitted for the MaxSAT Evaluation 2018. It uses a linear MaxSAT algorithm coupled with solution-based phase saving and a varying resolution approach. Our experimental results have shown that significant improvements could be achieved when compared with *maxroster*, one of the top performing solvers from the incomplete track last year. However, more detailed experimental results, such as those provided by the MaxSAT competition, are required to draw stronger conclusions.

References

[1] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2 system description," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010.

[2] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a modular maxSAT solver," in *Proceedings of SAT-14*, pp. 438–445.

[3] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proceedings of IJCAI'09*.

[4] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of DAC'01*, pp. 530–535.

[5] I. Abío Roig, "Solving hard industrial combinatorial problems with SAT," *Ph.D. thesis, Technical University of Catalonia (UPC)*, 2013.

[6] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Proceedings of CP'98* (M. Maher and J.-F. Puget, eds.), pp. 417–431, Springer.

[7] C. Ansótegui, F. Didier, and J. Gabàs, "Exploiting the structure of unsatisfiable cores in maxSAT," in *Proceedings of IJCAI-15*, pp. 283–289.

[8] E. Demirović and P. J. Stuckey, "Solution-based phase saving and large neighbourhood search," in *to appear in the proceedings of CP'18'*.

[9] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.

Loandra: Core-Boosted Linear Search Entering MaxSAT Evaluation 2019

Jeremias Berg*, Emir Demirović†, Peter Stuckey†‡
*HIIT, Department of Computer Science, University of Helsinki, Finland
†University of Melbourne, Australia
‡Data61, CSIRO, Australia

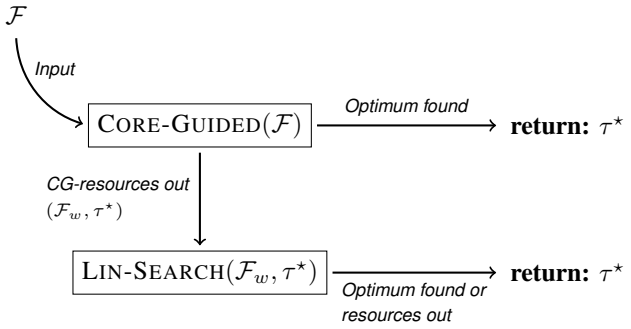

Fig. 1: The structure of Loandra.

I. Preliminaries

We briefly overview the Loandra MaxSAT-solver as it participated in the incomplete track of the 2019 MaxSAT Evaluation. A more thorough discussion can be found in [4]. We assume familiarity with conjunctive normal form (CNF) formulas and weighted partial maximum satisfiability (MaxSAT). Treating a CNF formula as a set of clauses a MaxSAT instance $\mathcal{F}$ consists of two CNF formulas, the hard clauses F_h and the soft clauses F_s, as well a weight w_c associated with each $C \in F_s$. A solution to $\mathcal{F}$ is an assignment τ that satisfies F_h. The cost of a solution τ is the sum of weights of the soft clauses falsified by τ. An optimal solution is one with minimum cost over all solutions. An unsatisfiable core κ of $\mathcal{F}$ is a subset of soft clauses s.t. $F_h \wedge \kappa$ is unsatisfiable.

II. Core-Boosted Linear Search

Loandra consists of two main components: CORE-GUIDED, a core-guided reformulation algorithm extended with stratification [1], and LIN-SEARCH, a SAT/UNSAT linear search algorithm. Both components make extensive use of Boolean Satisfiability (SAT) solvers. On input $\mathcal{F}$, CORE-GUIDED searches for an optimal MaxSAT solution by iteratively extracting unsatisfiable cores with a SAT solver and modifying a working instance (initialised to $\mathcal{F}$) in order to rule out them as sources of unsatisfiability. LIN-SEARCH iteratively queries the SAT solver for a solution of lower cost that the currently best known one. Both components are *complete*, i.e. given enough time and memory, both will compute an optimal solution. More importantly for the incomplete track, they also are *any-time*, i.e. both can output intermediate solutions during search. Note that stratification allows treating CORE-GUIDED as an any-time algorithm.

Figure 1 overviews the structure of Loandra. It uses core-boosting as described in [4] in order to exploit the strengths and alleviate the weaknesses of its individual components. On input $\mathcal{F}$ Loandra starts in a core-guided phase by invoking CORE-GUIDED on $\mathcal{F}$. If the optimal solution isn't found within the time allocated to the core-guided phase, the execution switches to a linear phase and LIN-SEARCH is invoked with $\tau^\star$, the best solution found by the core-guided phase, and $\mathcal{F}_w$, the final working instance of CORE-GUIDED. The linear search runs until either finding the optimal solution or reaching the time out, at which point the currently best known solution is returned.

III. Implementation details

The version of Loandra that entered the 2019 Evaluation is the same one as was experimented on in [4]. As the instantiation of CORE-GUIDED we use a reimplementation of the PMRES [8] MaxSAT algorithm extended with weight aware core extraction (WCE) [5] and clause hardening. The core-guided phase runs until no more cores can be found with the stratification bound set to 1, or 30s has passed.

As the instantiation of LIN-SEARCH we use a reimplementation of LinSBPS [3], SAT/UNSAT linear search extended with solution-based phase saving and varying resolution. Following LinSBPS we use the generalized totalizer encoding [6] to convert the PB constraints needed in linear search to CNF. All algorithms are implemented on top of the publicly available Open-WBO system [7] using Glucose 4.1 [2] as the back-end SAT solver.

IV. Compilation and Usage

Building and using Loandra resembles building and using Open-WBO. A statically linked version of Loandra in release mode can be built by running MAKE RS in the base folder.

After building, Loandra can be invoked from the terminal. Except for the formula file, Loandra accepts a number of command line arguments: the flag *-pmreslin-cglim* sets the maximum time that the core-guided phase can run for (in seconds). The rest of the flags resemble the flags accepted by Open-WBO; invoke ./loandra_static –help-verb for more information.

REFERENCES

[1] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy, "Improving SAT-based weighted MaxSAT solvers," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 7514. Springer, 2012, pp. 86–101.

[2] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proc IJCAI*. Morgan Kaufmann Publishers Inc., 2009, pp. 399–404.

[3] F. Bacchus, M. Järvisalo, R. Martins *et al.*, "Maxsat evaluation 2018," https://maxsat-evaluations.github.io/2018/, 2018, accessed: 2018-9-05.

[4] J. Berg, E. Demirovic, and P. Stuckey, "Core-boosted linear search for incomplete maxsat," in *Proc CPAIOR*, ser. Lecture Notes in Computer Science, vol. ???? Springer, 2019, p. ??? (to appear).

[5] J. Berg and M. Järvisalo, "Weight-aware core extraction in SAT-based MaxSAT solving," in *Proc. CP*, ser. Lecture Notes in Computer Science, 2017, to appear.

[6] S. Joshi, R. Martins, and V. M. Manquinho, "Generalized totalizer encoding for pseudo-boolean constraints," in *Proc. CP*, ser. LNCS, vol. 9255, 2015, pp. 200–209.

[7] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: A modular MaxSAT solver," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 8561. Springer, 2014, pp. 438–445.

[8] N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided MaxSAT resolution," in *Proc. AAAI*. AAAI Press, 2014, pp. 2717–2723.

Open-WBO-Inc in MaxSAT Evaluation 2019

Saurabh Joshi, Prateek Kumar, Sukrut Rao
{sbjoshi,cs15btech11031,cs15btech11036}@iith.ac.in
Indian Institute of Technology Hyderabad, India

Ruben Martins
rubenm@cs.cmu.edu
CMU, USA

I. Introduction

Open-WBO-Inc [1], [2] is developed on top of Open-WBO [3], [4], [5] and placed first and second on the weighted incomplete tracks for 60 and 300 seconds in the MaxSAT Evaluation 2018, respectively. For many applications that can be encoded into MaxSAT, it is important to quickly find solutions even though these may not be optimal. Open-WBO-Inc is designed to find a good solution[1] in a short amount of time. Since Open-WBO-Inc is based on Open-WBO, it can use any MiniSAT-like solver [6]. For this evaluation, we use Glucose 4.1 [7] as our back-end SAT solver.

II. Algorithms

For the MaxSAT Evaluation 2019, we restrict Open-WBO-Inc to the weighted category where it uses the novel approximation algorithms that have been recently proposed [1], [2]. In particular, we submitted two versions of Open-WBO-Inc: inc-bmo-complete and inc-bmo-satlike.

Both inc-bmo-complete and inc-bmo-satlike versions are based on bounded multilevel optimization [8] using a variant of linear search algorithm SAT-UNSAT [9]. The algorithms used in these versions consider n objective functions where n is the number of different weights in the MaxSAT instance. This is done by performing a sequence of calls to a SAT solver and refining an upper bound μ on the number of unsatisfied soft clauses. To restrict μ at each iteration, we need to encode cardinality constraints into CNF, for which, incremental Totalizer encoding [4] has been used. Once for a given objective function the upper bound μ cannot be improved, it is frozen, and the next objective function in the order is optimized.

If an optimal solution is found when using this algorithm, then it is not necessarily an optimal solution of the input formula. inc-bmo-complete and inc-bmo-satlike versions different between themselves once this occurs. inc-bmo-complete keeps the best-known solution and resumes the search using the LSU algorithm which can potentially find better solutions and prove optimality. In contrast, inc-bmo-satlike changes the search algorithm to SATLike [10], a MaxSAT stochastic algorithm. The best model found by the first phase is passed to SATLike as its initial starting model.

[1]By "good solution" we mean that it can be potentially suboptimal but is not far from the optimal solution.

III. Availability

We submit the source of Open-WBO-Inc as part of our submissions to the MaxSAT Evaluations 2019. inc-bmo-complete version and the full Open-WBO-Inc framework is available under a MIT license in GitHub at *https://github.com/sbjoshi/Open-WBO-Inc*. The inc-bmo-satlike version is available in the same repository under the *satlike* branch.

Acknowledgments

We would like to thank Laurent Simon and Gilles Audemard for allowing us to use Glucose in the MaxSAT Evaluation. We would also like to thank Vasco Manquinho, Inês Lynce, Mikoláš Janota, Miguel Terra-Neves and Norbert Manthey for their contributions to Open-WBO on which Open-WBO-Inc is based.

References

[1] S. Joshi, P. Kumar, R. Martins, and S. Rao, "Approximation Strategies for Incomplete MaxSAT," in *CP*. Springer, 2018.
[2] S. Joshi, P. Kumar, S. Rao, and R. Martins, "Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT," in *Journal on Satisfiability, Boolean Modeling and Computation*. IOS Press, 2019.
[3] R. Martins, V. Manquinho, and I. Lynce, "Open-WBO: a Modular MaxSAT Solver," in *SAT*, ser. LNCS, vol. 8561. Springer, 2014, pp. 438–445.
[4] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental Cardinality Constraints for MaxSAT," in *CP*. Springer, 2014, pp. 531–548.
[5] M. Neves, R. Martins, M. Janota, I. Lynce, and V. Manquinho, "Exploiting Resolution-Based Representations for MaxSAT Solving," in *SAT*. Springer, 2015, pp. 272–286.
[6] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*. Springer, 2003, pp. 502–518.
[7] G. Audemard and L. Simon, "Predicting Learnt Clauses Quality in Modern SAT Solvers," in *IJCAI*, 2009, pp. 399–404.
[8] J. Marques-Silva, J. Argelich, A. Graça, and I. Lynce, "Boolean lexicographic optimization: algorithms & applications," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3-4, pp. 317–343, 2011.
[9] D. Le Berre and A. Parrain, "The Sat4j library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, no. 2-3, pp. 59–6, 2010.
[10] Z. Lei and S. Cai, "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT," in *IJCAI*. ijcai.org, 2018, pp. 1346–1352.

SATLike3.0-c: solver description

Zhendong Lei and Shaowei Cai
State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing, China
{leizd,caisw}@ios.ac.cn

***Abstract*—In this document, we briefly describe the SATLike3.0-c solver participating in MaxSAT competition 2019.**

I. Introduction

SATLike3.0-c participates in Incomplete Track. It has two engines, one is local search solver SATLike [1] and another is LinSBPS solver. The main solver is SATLike, and LinSBPS is called only when SATLike fails to find a feasible solution withine 40 seconds, which does not usually happen. Note that if SATLike finds a feasible solution within 40 seconds, then it continues to improve the solution and would not call LinSBPS at all. We use an improved version of SATLike namely SATLike 3.0.

II. Implementation

SATLike3.0-c can be directly used to solve both PMS and WPMS. The only difference is the parameters setting.

References

[1] Z. Lei and S. Cai, "Solving (weighted) partial maxsat by dynamic local search for SAT," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 2018, pp. 1346–1352.

`sls-mcs` and `sls-lsu`: Description

Andreia P. Guerreiro[1], Miguel Terra-Neves[1,2], Inês Lynce[1], José Rui Figueira[3], and Vasco Manquinho[1]

[1]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{andreia, ines, vmm}@sat.inesc-id.pt
[2]OutSystems, Portugal
miguel.neves@outsystems.com
[3]CEG-IST, Instituto Superior Técnico, Universidade de Lisboa, Portugal
figueira@tecnico.ulisboa.pt

1 Introduction

We developed a solver that integrates SAT-based techniques in a Stochastic Local Search (SLS) solver for MaxSAT. In our solver, the control of the solving process changes from SAT-based procedures to stochastic procedures and vice-versa. At each step, each procedure tries to build upon the information received from the other, instead of being independent procedures. The idea is to use the SLS solver as the main procedure, and occasionally, use an unsatisfiability-based algorithm to correct the SLS current (unsatisfiable) assignment into a satisfiable one, and use a procedure based on Minimal Correction Subset (MCS) enumeration to improve the current solution. We submitted two versions of the new solver for the unweighted incomplete MaxSAT track, and two version for the weighted incomplete MaxSAT track.

2 Using SAT Techniques in Local Search

One of the shortcomings of SLS algorithms is that these solvers have difficulties in dealing with highly constrained formulas. Therefore, it might be the case that the SLS algorithm is unable to satisfy the set of hard clauses or gets stuck in some local minima. In these cases, using SAT-based techniques to find a satisfiable assignment would be beneficial.

2.1 Assignment Correction

Consider the case when the SLS algorithm is unable to change from an unsatisfiable assignment ν into a better assignment. Our solver performs a correction to ν in order to guide the SLS algorithm to the feasible region of the search space. First, we start by building a set of assumption literals corresponding to the assignment ν. Next, a SAT call on the set of hard clauses, ϕ_h, is made. Clearly, if ν is not feasible, then this call returns `UNSAT` and returns an unsatisfiable core. The assumption literals that occur in such an unsatisfiable core are removed from the set of assumptions, and a new SAT call is made. The same procedure is repeated until a satisfiable assignment is found.

A conflict limit is defined for the correction procedure. If the conflict budget is not enough to find a satisfiable assignment, then our algorithm applies a similar procedure with a more aggressive strategy where at each iteration 50% of the literals in the set of assumptions are removed. Since the correction procedure only depends on the hard clauses, there is no guarantee regarding its quality. As a result, we also apply a SAT-based improvement procedure.

2.2 Assignment Improvement

Given a MaxSAT instance ϕ, a set of assumptions $\mathcal{A}$, a satisfiable assignment ν, and conflict budget, the goal of this assignment improvement algorithm is to find a better quality solution for ϕ through an MCS enumeration procedure.

The algorithm starts by building a working for-

mula from the set of hard clauses ϕ_h and the set of assumptions $\mathcal{A}$. Next, the algorithm iterates over all MCSes of ϕ, constrained to the set of assumptions $\mathcal{A}$ and returns the best assignment found. Each time a new MCS is found, a blocking clause is added to prevent the enumeration of the same MCS later on. The algorithm returns the best solution found before the conflict budget runs out. Note that the set of literals $\mathcal{A}$ restricts the MCS enumeration procedure. This results in a localized MCS enumeration.

Many different improvement procedures can be devised, including the usage of complete methods. For example, an alternative is to replace the MCS enumeration algorithm by a call to a Linear Sat-Unsat algorithm (LSU). The call to the LSU algorithm is also limited to a number of conflicts, and all literals in $\mathcal{A}$ are forced to be satisfied. Hence, the LSU call is also restricted to a localized region of the search space.

3 Incomplete Track

As SATLike [2], an SLS algorithm, was one of the best performing solvers in the incomplete solver track in the MaxSAT Evaluation 2018, we used SATLike in our implementation.

3.1 Unweighted Instances

Two solvers were submitted for the unweighted incomplete track: `sls-mcs` and `sls-lsu`. In `sls-mcs`, the `SATLike` solver[1] is extended with the assignment correction algorithm and the assignment improvement algorithm based on MCS enumeration. The difference from `sls-mcs` to `sls-lsu` is on the assignment improvement algorithm. In `sls-lsu`, the linear sat-unsat assignment improvement algorithm is used.

Both `sls-mcs` and `sls-lsu` use the Glucose SAT solver (version 4.1) on the assignment correction procedure. Moreover, the CLD [3] algorithm is used as the MCS algorithm in `sls-mcs`. The linear sat-unsat algorithm used in `sls-lsu` is an adapted version of the one available at the `open-wbo` open source MaxSAT solver. The conflict limits of the correction and the improvement algorithms were set to 10^5. In both `sls-mcs` and `sls-lsu`, the assignment correction/improvement algorithm is called when SATlike has reached half of the maximum number of iterations without improvement. In such a case, the correction algorithm is called if the current assignment ν does not satisfy all hard clauses, otherwise the improvement algorithm is directly called with approximately half of the literals in the current assignment ν as assumptions. These assumption literals are randomly chosen from ν.

[1]The source code of `SATLike` is publicly available at the 2018 MaxSAT evaluation `https://maxsat-evaluations.github.io/2018/descriptions.html`

3.2 Weighted Instances

Two versions of the solver were submitted for the weighted incomplete track: `sls-mcs` and `sls-mcs2`. In both versions, the stratified CLD algorithm [4] is used as the MCS algorithm. Unlike `sls-mcs`, `sls-mcs2` does not consider the assumptions $\mathcal{A}$ as hard clauses in the MCS enumeration procedure.

Acknowledgments

We would like to thank Zhendong Lei and Shaowei Cai for making the code of SATLike available online.

References

[1] Lang, J. (ed.): Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. ijcai.org (2018)

[2] Lei, Z., Cai, S.: Solving (weighted) partial maxsat by dynamic local search for SAT. In: Lang [1], pp. 1346–1352

[3] Marques-Silva, J., Heras, F., Janota, M., Previti, A., Belov, A.: On Computing Minimal Correction Subsets. In: International Joint Conference on Artificial Intelligence. pp. 615–622 (2013)

[4] Terra-Neves, M., Lynce, I., Manquinho, V.M.: Stratification for constraint-based multi-objective combinatorial optimization. In: Lang [1], pp. 1376–1382

TT-Open-WBO-Inc: Tuning Polarity and Variable Selection for Anytime SAT-based Optimization

Alexander Nadel
Email: alexander.nadel@hotmail.com

***Abstract*—This document is a description of the solver TT-Open-WBO-Inc, submitted to the weighted incomplete tracks of MaxSAT Evaluation 2019. We tuned the polarity and variable selection strategies of the underlying SAT solver in the best-performing MaxSAT solver in the Weighted-Incomplete-60-Second track of MaxSAT Evaluation 2018 – Open-WBO-Inc-BMO [6], [7].**

I. Introduction

The main goal of this submission is to experiment with a new polarity selection heuristic and an enhancement to the variable decision strategy for SAT-based anytime Weighted MaxSAT solving in the state-of-the-art algorithm Open-WBO-Inc-BMO [6], [7]. In principle, our heuristics can be applied to solving any optimization problem with a SAT-based anytime algorithm.

We call our polarity selection heuristic Target-Optimum-Rest-Conservative (TORC) and the enhancement to the variable selection strategy Target-Score-Bump (TSB). Our heuristics are detailed in [9]. We provide a brief (yet precise) description in this document.

II. Preliminaries

A Weighted MaxSAT instance comprises a set of *hard* satisfiable clauses H and a set of weighted *soft* constraints $T = \{t_{n-1}, t_{n-2}, \ldots, t_0\}$, where each constraint t_i is associated with a strictly positive integer weight w_i. The *weight of a variable assignment* μ is $\mathit{unsWt}(T, \mu) = \sum_{i=0}^{n-1} \neg\mu(t_i) * w_i$, that is, the overall weight of T's bits, falsified by μ. Given a Weighted MaxSAT instance, a Weighted MaxSAT solver is expected to return a model having the minimum possible weight. For the rest of this document, for convenience and without restricting generality, it is assumed that every soft constraint is a unit clause (that is, a clause containing one literal). An arbitrary soft constraint t_i, reducible to a set of clauses F, can be transformed to a unit clause s', where s' is a fresh variable, by adding the clause $\neg s' \vee c$ to H for each clause $c \in F$. Thus, T can be thought of as a bit-vector, where t_0 is its Least Significant Bit (LSB) and t_{n-1} is its Most Significant Bit (MSB). T is called the *target bit-vector*, or, simply, the *target* and every $t_i \in T$ is called a *target bit*.

Recall that modern SAT solvers apply phase saving [11] as their polarity selection heuristic. In phase saving, once a variable is picked by the variable decision heuristic, the literal is chosen according to its latest value, where the values are normally initialized with 0.

It turned out that overriding phase saving in the context of anytime SAT-based optimization algorithms, which generate an improving set of models $\{\mu_1, \mu_2, \ldots, \mu_n\}$ over time, is advantageous. In this context, one can distinguish between the optimistic and the conservative approaches to polarity selection. The optimistic approach [3], [5], [10] sets the polarity of the target bits to 1; it works well when the actual solution is close to the optimum. The conservative approach [1], [4], [12] sets the polarity of all the variables (or all the original variables) to the previous best solution.

III. Target-Optimum-Rest-Conservative (TORC) Polarity Selection

We propose a new polarity selection heuristic, which we call *Target-Optimum-Rest-Conservative (*TORC*)*.

Before the initial SAT invocation, TORC *fixes* the polarity of all the *target* variables to the optimal value. Then, after each new improving model μ_i is encountered, the polarity of *all* the *non-target* variables are fixed to their values in μ_i.

In other words, whenever the variable decision heuristic chooses:

1) A target variable: TORC sets its polarity to 1 (to be optimistic).
2) A non-target variable: TORC sets its polarity to its value in the best model so far (to be conservative; only after the first SAT invocation is completed)

Note that, after the initial SAT call, TORC sets the polarity every single time a new decision variables is picked.

TORC has been designed to leverage the best of both the conservative and the optimistic worlds. On one hand, we are interested in taking advantage of the conservative heuristic, which is known to find the next improved model more quickly than the default heuristic by looking near the previous model. At the same time, however, we would like to encourage the values of the target variables to be as close to the optimum as possible in order to move more quickly towards the optimum.

IV. Target Score Bump (TSB)

We would like to experiment with tuning the SAT solver's variable selection heuristic for anytime SAT-based optimization.

Modern SAT solvers mostly use variants of the VSIDS variable decision heuristic [8]. VSIDS associates a score with every variable and picks as the next decision the variable with the greatest score. Open-WBO-Inc-BMO uses Glucose 4.1 SAT

solver [2]. Glucose 4.1 has a function *varBumpActivity(v,b)*, which bumps up the score of variable v by b.

Our proposed heuristic–`Target-Score-Bump (TSB)`–bumps up the variable scores of the *target bit* variables, so as to improve their chances of being picked early. We would also like to give some preference to target bits having greater weight.

We apply `TSB` prior to the initial SAT invocation as follows.

Let the minimal target-bit weight be $\min = \min(w_0, \ldots, w_{n-1})$ and the maximal target-bit weight be $\max = \max(w_0, \ldots, w_{n-1})$. For every variable t of target bit literal t_i of weight w_i, we apply the function *varBumpActivity(t,b)*, where b is

$(w_i - \min)/(\max - \min) * weightBump + varBump$.

Both weightBump and varBump are user-given parameters. They regulate the relative importance of the weight in the scores. The default version of `TT-Open-WBO-Inc` uses weightBump = 113 and varBump = 552.

REFERENCES

[1] C. Ansótegui and J. Gabàs. WPM3: an (in)complete algorithm for weighted partial maxsat. *Artif. Intell.*, 250:37–57, 2017.

[2] G. Audemard and L. Simon. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27(1):1–25, 2018.

[3] D. L. Berre and A. Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[4] E. Demirovíc, G. Chu, and P. J. Stuckey. Solution-based phase saving for CP: A value-selection heuristic to simulate local search behavior in complete solvers. In *CP 2018*, pages 99–108, 2018.

[5] E. Di Rosa and E. Giunchiglia. Combining approaches for solving satisfiability problems with qualitative preferences. *AI Commun.*, 26(4):395–408, 2013.

[6] S. Joshi, P. Kumar, V. Manquinho, R. Martins, A. Nadel, and S. Rao. Open-WBO-Inc in MaxSAT evaluation 2018. volume B-2018-2 of *Department of Computer Science Series of Publications B*, page 16. University of Helsinki, 2018.

[7] S. Joshi, P. Kumar, R. Martins, and S. Rao. Approximation strategies for incomplete maxsat. In *CP 2018*, pages 219–228, 2018.

[8] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, pages 530–535, 2001.

[9] A. Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *FMCAD 2019*, 2019. To appear.

[10] A. Nadel and V. Ryvchin. Bit-vector optimization. In *TACAS 2016*, pages 851–867, 2016.

[11] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.

[12] I. A. Roig. *Solving hard industrial combinatorial problems with SAT*. Dissertation, Polytechnic University of Catalonia, 2013.

BENCHMARK DESCRIPTIONS

MaxSAT Evaluation 2019: Benchmark Sets and Selection Procedure

Fahiem Bacchus
University of Toronto
Canada

Matti Järvisalo
University of Helsinki
Finland

Ruben Martins
Carnegie Mellon University
USA

I. Introduction

We give a brief description of the benchmark sets use in MaxSAT Evaluation 2019 and details on how these benchmarks were selected

II. New Benchmark Domains

Based on an open call for benchmarks, we obtained a total of 25,788 new benchmarks encoding instances from six different problem domains.

- Minimum weight dominating set problem (10 benchmarks)
- Identification of security-critical cyber-physical components in weighted AND/OR graphs (80 benchmarks)
- Consistent query answering (19 benchmarks)
- MaxSAT queries in the design of interpretable rule-based classifiers (17,135 benchmarks)
- Maximum common sub-graph extraction (8,544 benchmarks)
- Parametric RBAC maintenance via MaxSAT (882 benchmarks)

III. Overview of MSE19 Benchmark Sets

The MaxSAT Evaluation 2019 benchmark sets were constructed by selecting instances both from the set of instances previously obtained and used in previous MaxSAT Evaluations and from the set of new benchmarks submitted for 2019.

The benchmark selection procedure, detailed in the next section, resulted in the following benchmark sets for the 2019 evaluation.

Complete track

- Unweighted (599 benchmarks):
  - 48 families of benchmarks
  - 201 benchmarks were used in MSE18
  - 66 benchmarks are new
  - 332 benchmarks were previously submitted
- Weighted (583 benchmarks):
  - 39 families of benchmarks
  - 191 benchmarks were used in MSE18
  - 97 benchmarks are new
  - 295 benchmarks were previously submitted

Incomplete track

- Unweighted (299 benchmarks):
  - 112 benchmarks were used in MSE18
  - 60 benchmarks are new
  - 127 benchmarks were previously submitted
- Weighted (297 benchmarks):
  - 97 benchmarks were used in MSE18
  - 37 benchmarks are new
  - 163 benchmarks were previously submitted

IV. Benchmark Selection

This year we implemented a new method for selecting the evaluation test set. For the evaluations we have collected 7,438 instances from prior competitions as well as the 25,788 new instances collected this year. These benchmarks are broken up into 51 different families containing unweighted instances, and 42 different families containing weighted instances.

Given the size of this set, this year we developed a new scheme for selecting a more randomized and representative set of benchmarks for the evaluation. Our aim was to select 600 benchmark instances for both the weighted and unweighted complete tracks along with 300 instances in each of the four weighted/unweighted, 60/300 second incomplete tracks.

The first step is an attempt to exclude "uninformative" instances. We decided to exclude the following types of instances before selecting randomly from the remaining instances.

1) In 2017 we decided to remove the random track from the evaluation due to difficulties in finding interesting new benchmarks. In reviewing the remaining families (those previously classified as industrial and crafted) we found an additional nine families that were in fact random instances. These included maxcut and maxclique instances on random graphs, some random maxone SAT problems, and random set cover problems.
2) We excluded all instances that we had found to have zero optimum. Such instances do not require optimization as a SAT solver can find a solution to all of the clauses (both hard and soft). In fact, during the evaluation we were able to find three additional instances with a zero optimum. These instances will be removed from future evaluations.
3) Some recently submitted instances were very large with the input files being almost 5GB large. Running these on some different MaxSat solvers showed that they were

not solvable within the time and space constraints of the evaluation. In all 22 instances were excluded as being too large.

4) We had data on the previous 7,438 benchmarks, including run times of three quite different MaxSat solvers MaxHS, RC-2, and QMaxSat. All instances that could be solved by all three solvers in less than 10 seconds were excluded as being too easy.
5) Among the new instances we did not as yet have extensive data. But we did run one MaxSat solver (MaxHS) on all of these instances (with a low time bound), and excluded all instances that were solved in less than 0.25 seconds. In future evaluations we should be able to get data from other solvers on these instances and will be able to apply the previous test for instances being too easy.
6) For the incomplete tracks we tried to select instances that were not solvable by any exact solver within the time bound of the track. Thus, for the 60 second track we tried to select instances that took exact solvers more than 60 seconds, and similarly for the 300 second track. During the evaluation we did find that some of these instances could be solved exactly in less time that the track's limit. So in future evaluations these instances will be removed from the incomplete track.

After excluding the above instances, we wanted to select a representative evaluation suite for each track. Since collecting new benchmark families requires some effort, we wanted to highlight such families by selecting more instances from them. Hence the random selection of instances was performed in three phases.

First, we tried to make each new family be 5% of the total evaluation suite. We decided that having more than 5% of the evaluation suite be from a single family would introduce too much bias into the evaluation suite, so we set the limit at 5%. We had 4 new weighted and 4 new unweighted families submitted this year. So, e.g., since the evaluation suite for the complete unweighted track had a target of 600 instances we tried to select 30 instances (5% of 600) from each of these new families. If any of these families contained less than 30 non-excluded instances we selected all of the them. From each new family with more than 30 non-excluded instances we selected 30 instances randomly using the scheme described below.

Second, once we had selected as many instances as we could, up to the 5% limit from the new families, we filled out the suite with instances from the old benchmark families. For example, for the unweighted complete track we could only select 66 instance from the new families. Hence, we had to select the remaining 534 instances (i.e., 600-66) from the old families. This was done by first drawing from a multinomial distribution with equal probabilities. The drawn multinomial gives the number of instances to select from each family. Continuing our example, there were 47 old unweighted families. Hence using the multinomial distribution we select a random tuple of 47 numbers that sums to 534. These 47 numbers tell us how many instances to draw from each of the 47 families. From the properties of the multinomial distribution we expect a fairly even distribution of number of instances from the individual families. However, it is very unlikely that we will select exactly the same number of instances from each family. This multinomial step adds some randomness to the selected suite while retaining good representation from the different families.

There are two slight complications however. First, the random multinomial tuple might tell us to draw more instances from a family than exist. For example, the multinomial might specify drawing 15 instances from a family that contains only 6 non-excluded instances. In this case, we would select all 6 instances from that family. However, this would leave us with 9 fewer instances than we wanted to select. We addressed this issue by doing repeated rounds of multinomial selection until we obtained the number of instances we wanted to have in the suite. In particular, in each round draw a new multinomial tuple, this time over the non-exhausted families, that sums to the number of instances we still needed to complete the evaluation suite. Thus, each round gets us closer to completing the evaluation suite, and we only needed 2-3 rounds.

The second complication is that some families are actually broken up into sub-families. We solve this problem by recursively applying the multinomial technique. For example, suppose we wanted to select 20 instances from family F, and F had 3 sub-families. We would then draw a triple of numbers from the multinomial distribution that sums to 20. This triple of numbers tells us how many instances to draw from each sub-family so that a total of 20 instances come from F. By applying this technique recursively we could also handle families that had sub-sub-families, although this did not happen in our case.

Given that the previous stages determined that we needed to select k instances from family F the final stage determined how we select them. Selecting from the instances of F uniformly is not reasonable since the difficulty of the instances in F is often uniformly distributed. Instead we first partitioned the instances in F into quintiles based on their size, where the size of an instance was taken to be the sum of the lengths of all of its clauses (both hard and soft). To select k instances from F, we then drew a random 5-tuple from the multinomial distribution that summed to k. This multinomial tuple tells us how many instances to draw from each quintile. The final draw of instances from each quintile was done uniformly at random without replacement.[1]

[1]Using a multinomial to decide how many instances to draw from each quintile is probably not needed; drawing a equal number from each quintile would have been sufficient. However, it was easy to do this given the software we had written to do the previous stages, and the extra randomness might be useful.

Encoding Consistent Query Answering to MaxSAT

Akhil A. Dixit
UC Santa Cruz
Santa Cruz, USA
akadixit@ucsc.edu

Phokion G. Kolaitis
UC Santa Cruz and IBM Research - Almaden
Santa Cruz, USA
kolaitis@ucsc.edu

***Abstract*—This document contains the brief description about the Weighted MaxSAT problem instances submitted to the MaxSAT Evaluation 2019.**

I. Original Problem Domain

All of the MaxSAT instances submitted for the evaluation originally come from the problem *Consistent Query Answering (CQA)* from relational databases. This problem was first studied by Arenas, Bertossi, and Chomicki in [1]. The background on this field of study, the necessary notions, and the formal description of the problem of CQA can be found in Section 2 of our paper "A SAT-based System for Consistent Query Answering", which has been accepted as a long-paper at the SAT 2019 conference. This paper contains four reductions from the complement of CQA to Boolean SAT and Weighted MaxSAT. We do not submit all instances generated during the experiments from our paper; we submit only 19 of them. This is due to the limit on the size of an email attachment.

II. Parameters Used to Generate Instances

A. Instances generated from synthetic databases

The Weighted MaxSAT instances from synthetic databases are generated using Reduction 2 and Algorithm 1 from our paper. The synthetic data used to produce the submitted instances were generated using the parameters listed in Table I. Since Algorithm 1 is iterative and it modifies the Weighted MaxSAT instance in each iteration, we submit the instances at the time of the termination of the algorithm (i.e., when the algorithm has found all variables in the instance which can be set to true in some satisfying assignment).

TABLE I
Parameters used to generate synthetic data

Parameter	Value
Number of tuples per relation	500,000
Size of inconsistent key-equal groups	2
Degree of inconsistency	10%
Degree of join	15%

The naming convention used for these instances is as follows. The name of each .wcnf file is of the form "synthetic-x.wcnf", where x is an integer corresponding to the database query used to generate the instance. For example, "synthetic-3.wcnf" was generated using the query q_3. The conjunctive queries used to generate these instances are divided into three categories based on the complexity of computing their consistent answers, namely, FO-rewritable, in P but not FO-rewritable, and coNP-complete. The precise definitions of the queries follow.

FO-rewritable consistent answers:

$q_2(z,w) := \exists x,y,v\ (R_1(\underline{x},y,z) \wedge R_2(\underline{y},v,w))$
$q_3(z) := \exists x,y,v,u,d\ (R_1(\underline{x},y,z) \wedge R_3(\underline{y},v) \wedge R_2(\underline{v},u,d))$
$q_5(z) := \exists x,y,v,w\ (R_1(\underline{x},y,z) \wedge R_4(\underline{y,v},w))$
$q_6(z) := \exists x,y,x',w,d\ \ (R_1(\underline{x},y,z) \wedge R_2(\underline{x'},y,w) \wedge R_5(\underline{x},y,d))$
$q_7(z) := \exists x,y,w,d\ (R_1(\underline{x},y,z) \wedge R_2(\underline{y},x,w) \wedge R_5(\underline{x},y,d))$

In P, but not FO-rewritable, consistent answers:

$q_8(z,w) := \exists x,y\ (R_1(\underline{x},y,z) \wedge R_2(\underline{y},x,w))$
$q_{10}(z,w,d) := \exists x,y,u\ \ (R_1(\underline{x},y,z) \wedge R_2(\underline{y},x,w) \wedge R_4(\underline{y},u,d))$
$q_{12}(v,d) := \exists x,y,z,u\ (R_3(\underline{x},y) \wedge R_6(\underline{y},z) \wedge R_1(\underline{z},x,d) \wedge R_4(\underline{x,u},v))$
$q_{13}(v) := \exists x,y,z,u\ \ (R_3(\underline{x},y) \wedge R_6(\underline{y},z) \wedge R_7(\underline{z},x) \wedge R_4(\underline{x,u},v))$
$q_{14}(d) := \exists x,y,z,u\ \ (R_3(\underline{x},y) \wedge R_6(\underline{y},z) \wedge R_1(\underline{z},x,d) \wedge R_7(\underline{x},u))$

coNP-complete consistent answers:

$q_{15}(z) := \exists x,y,x',w\ (R_1(\underline{x},y,z) \wedge R_2(\underline{x'},y,w))$
$q_{18}(z,w) := \exists x,y,x',u,d\ \ (R_1(\underline{x},y,z) \wedge R_2(\underline{x'},y,w) \wedge R_4(\underline{y},u,d))$
$q_{19}(z,w,d) := \exists x,y,x',u\ \ (R_1(\underline{x},y,z) \wedge R_2(\underline{x'},y,w) \wedge R_4(\underline{y},u,d))$
$q_{20}(z) := \exists x,y,x',w,u,d,v\ \ (R_1(\underline{x},y,z) \wedge R_2(\underline{x'},y,w) \wedge R_4(\underline{y},u,d) \wedge R_3(\underline{u},v))$
$q_{21}(z,w) := \exists x,y,x',u,d,v\ \ (R_1(\underline{x},y,z) \wedge R_2(\underline{x'},y,w) \wedge R_4(\underline{y},u,d) \wedge R_3(\underline{u},v))$

B. Instances generated from the real-world data

The Weighted MaxSAT instances from the real-world database are generated using Reduction 4 and Algorithm 1 from our paper. The data used for these instances are about the food and safety inspections of restaurants based in New York and Chicago, and are taken from [2], [3]. Table II summarizes the properties of the real-world data.

The naming convention used for these instances is as follows. The name of each .wcnf file is of the form "real-x.wcnf", where x is an integer corresponding to the database

TABLE II
PROPERTIES OF THE REAL-WORLD DATA

Parameter	Value
Number of tuples per relation	From 31,100 to 229,000
Size of inconsistent key-equal groups	From 2 to 50
Degree of inconsistency	From 0% to 25%
Degree of join	From 0.5% to 30%

query used to generate the instance. For example, "real-3.wcnf" was generated using the query Q_3. The database schema and the set of integrity constraints (primary keys and functional dependencies) are given in Table 4 of our paper. The definitions of the unions of conjunctive queries used to generate the instances follow.

$$Q_2(x) := \exists y, z, w, v, y', z', w', v' \ (\text{NY\_Rest}(x, y, z, w, v) \wedge \text{CH\_Rest}(x, y', z', w', v'))$$

$$Q_3(x) := \exists y, z, w, v, y', z', w', v', q, r, s, t, q', s', t' \ (\text{NY\_Rest}(x, y, z, w, v) \wedge \text{CH\_Rest}(x, y', z', w', v') \wedge \text{NY\_Insp}(y, q, r, s, t) \wedge \text{CH\_Insp}(y', q', r, s', t'))$$

$$Q_5(x) := \exists y, z, w, v, q, r, s \ (\text{CH\_Rest}(x, y, z, w, v) \wedge \text{CH\_Insp}(y, q, r, s, \text{'Fail'})) \cup \exists y, z, w, v, q, r, s \ (\text{NY\_Rest}(x, y, z, w, v) \wedge \text{NY\_Insp}(y, q, r, s, \text{'Fail'}))$$

$$Q_6(x, v) := \exists y, z, w, y', z', w', v', q, r, s \ (\text{CH\_Rest}(x, y, z, w, v) \wedge \text{NY\_Rest}(x, y', z', w', v') \wedge \text{NY\_Insp}(y', \text{'Not Critical'}, q, r, s))$$

REFERENCES

[1] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '99. New York, NY, USA: ACM, 1999, pp. 68–79. [Online]. Available: http://doi.acm.org/10.1145/303976.303983

[2] "Food Inspections, City of Chicago," Aug 2011. [Online]. Available: https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5

[3] "New York City Restaurant Inspection Results, Department of Health and Mental Hygiene (DOHMH)," Aug 2014. [Online]. Available: https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j

MaxSAT Evaluation 2019 - Benchmark: Identifying Security-Critical Cyber-Physical Components in Weighted AND/OR Graphs

Martín Barrère*, Chris Hankin*, Nicolas Nicolaou†, Demetrios G. Eliades†, Thomas Parisini‡
*Institute for Security Science and Technology, Imperial College London, UK
{m.barrere, c.hankin}@imperial.ac.uk
†KIOS Research and Innovation Centre of Excellence, University of Cyprus
{nicolasn, eldemet}@ucy.ac.cy
‡Department of Electrical and Electronic Engineering, Imperial College London, UK
{t.parisini}@imperial.ac.uk

***Abstract*—This paper presents a MaxSAT benchmark focused on identifying critical nodes in AND/OR graphs. We use AND/OR graphs to model Industrial Control Systems (ICS) as they are able to semantically grasp intricate logical interdependencies among ICS components. However, identifying critical nodes in AND/OR graphs is an NP-complete problem. We address this problem by efficiently transforming the input AND/OR graph-based model into a weighted logical formula that is then used to build and solve a Weighted Partial MAX-SAT problem. The benchmark includes 80 cases with AND/OR graphs of different size and composition as well as the optimal cost and solution for each case.**

I. Problem Overview

Over the last years, Industrial Control Systems (ICS) such as water treatment plants and energy facilities have become increasingly exposed to a wide range of cyber-physical threats, having massive destructive consequences. Our work is focused on security metrics and techniques that can be used to measure and improve the security posture of ICS environments [1]. We use AND/OR graphs to model these systems as they allow more realistic representations of the complex interdependencies among cyber-physical components that are normally involved in real-world settings [2], [3]. In that context, we have designed a security metric, detailed in [1], whose objective is to identify the set of critical AND/OR nodes (ICS network components) that must be compromised in order to disrupt the operation of the system, with minimal cost for the attacker.

From a graph-theoretical perspective, our security metric looks for a minimal weighted vertex cut in AND/OR graphs. This is an NP-complete problem as shown in [3], [4], [5]. While well-known algorithms such as Max-flow Min-cut [6] and variants of it could be used to estimate such metric over OR graphs in polynomial time, their use for general AND/OR graphs is not evident nor trivial as they may fail to capture the underlying logical semantics of the graph. In that context, we take advantage of state-of-the-art MaxSAT techniques to address our problem.

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant No 739551 (KIOS CoE).

II. Simple Example

Let us consider a simple ICS network whose operational dependencies are represented by the AND/OR graph shown in Figure 1. The graph reads as follows: the actuator $c1$ depends on the output of software agent d. Agent d in turn has two alternatives to work properly; it can use either the readings of sensor a and the output from agent b together, or the output from agent b and the readings of sensor c together. In addition, each cyber-physical component has an associated attack cost that represents the effort required by an attacker to compromise that component. Now, considering these costs, the question we are trying to answer is: which nodes should be compromised in order to disrupt the operation of actuator $c1$, with minimal effort (cost) for the attacker? In other words, what is the least-effort attack strategy to disable actuator $c1$?

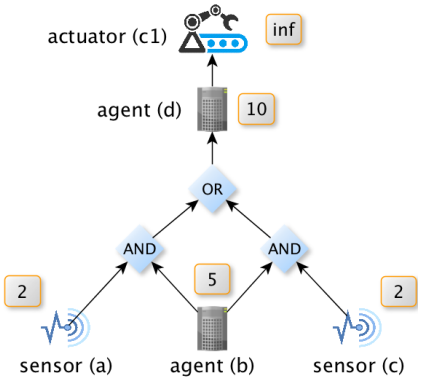

Fig. 1. AND/OR graph with sensors, software agents and actuators

Our example involves many attack alternatives, however, only one is minimal. The optimal strategy is to compromise nodes a and c with a total cost of 4. The compromise of these sensors will disable both AND nodes and consecutively the OR node, which in turn will affect node d and finally node $c1$.

III. MAX-SAT FORMULATION STRATEGY

Given a target node t, the input graph G can be used as a map to decode the dependencies that node t relies on. Therefore, G can be traversed backwards in order to produce a propositional formula that represents the different ways in which node t can be fulfilled. We call this transformation $f_G(t)$. In our example, $f_G(c1)$ is as follows:

$$f_G(c1) = c1 \wedge (d \wedge ((a \wedge b) \vee (b \wedge c)))$$

The goal of the attacker, however, is precisely the opposite, i.e., to disrupt node $c1$ somewhere along the graph. Therefore, we are actually interested in satisfying $\neg f_G(c1)$, which describes the means to disable $c1$. After applying a few logical rules, the conjunctive normal form (CNF) of $\neg f_G(c1)$ is:

$$\neg f_G(c1) = (\neg c1 \vee \neg d \vee \neg a \vee \neg b) \wedge (\neg c1 \vee \neg d \vee \neg b \vee \neg c)$$

In practice, we do not use the naive CNF conversion approach since it might lead to exponential computation times over large graphs. Instead, we use the Tseitin transformation [7], which can be done in polynomial time and essentially produces a new formula in CNF that is not strictly equivalent to the original formula (because there are new variables) but is equisatisfiable. This means that, given an assignment of truth values, the new formula is satisfied if and only if the original formula is also satisfied. Under that perspective, a logical assignment such that $\neg f_G(t) = true$ will indicate which nodes must be compromised (i.e. logically falsified) in order to disrupt the operation of the system.

Considering the CNF formula produced by the Tseitin transformation and a cost function $\varphi(n)$ that indicates the attack cost of a node n, we model our problem as a Weighted Partial MAX-SAT problem [8]. Hard clauses are essentially the clauses within the CNF formula:

$$\neg c1 \vee \neg d \vee \neg a \vee \neg b \qquad \neg c1 \vee \neg d \vee \neg b \vee \neg c$$

whereas soft clauses correspond to each atomic node in the graph with their corresponding penalties (costs) as follows:

a	b	c	d	$c1$
$\varphi(a) = 2$	$\varphi(b) = 5$	$\varphi(c) = 2$	$\varphi(d) = 10$	$\varphi(c1) = inf$

Therefore, a MAX-SAT solver will try to minimise the number of falsified variables as well as their weights, which in our problem equals to minimise the compromise cost for the attacker. Note: the additional variables introduced by the Tseitin transformation have cost/weight 0 in the formulation.

IV. AND/OR GRAPH GENERATION

The benchmark presented in this paper relies on META4ICS, a Java-based security metric analyser for ICS [1], [9]. We have used META4ICS to produce and analyse synthetic pseudo-random AND/OR graphs of different size and composition. To create an AND/OR graph of size n, we first create the target node. Afterwards, we create a predecessor which has one of the three types (atomic, AND, OR) according to a probability given by a compositional configuration predefined for the experiment. For example, a configuration of $(60, 20, 20)$ means 60% of atomic nodes, 20% of AND nodes and 20% of OR nodes. We repeat this process creating children on the respective nodes until we approximate the desired size of the graph n. Node costs, represented by $\varphi(n)$, are integer values randomly selected between 1 and 100.

The benchmark also includes the solutions obtained by META4ICS for each case, including resolution time, total cost and critical nodes. Currently, META4ICS uses SAT4J [10] and a Python-based linear programming approach as MaxSAT solvers. The tool runs all available solvers in parallel and picks the first one that comes up with a valid solution.

V. BENCHMARK DESCRIPTION

Out dataset includes 80 cases in total, and can be obtained at [9]. There are four different sizes of AND/OR graphs involving 5000, 10000, 15000, and 20000 nodes (20 cases each). For each graph size, we consider two different graph configurations, 80/10/10 and 60/20/20, which determine the composition of the graphs (10 cases each). Table I shows the identifiers of the cases within each one of these categories.

Nodes/Configurations	**80/10/10 config**	**60/20/20 config**
5000	1 to 10	11 to 20
10000	21 to 30	31 to 40
15000	41 to 50	51 to 60
20000	61 to 70	71 to 80

TABLE I
BENCHMARK CASES AND CONFIGURATIONS

Each case is specified in an individual **.wcnf** (DIMACS-like, weighted CNF) file named with the case id and the number of nodes involved. The weight for hard clauses (*top* value) has been set to 1.0×10^6. Tables II and III detail each case as well as the results obtained with our tool. The field **id** identifies each case; **gNodes** indicates the total number of nodes in the original AND/OR graph; **gAT**, **gAND** and **gOR** indicate the approximate composition of the graph in terms of atomic (cyber-physical components), AND and OR nodes; **tsVars** and **tsClauses** show the number of variables and clauses involved in the MaxSAT formulation after applying the Tseitin transformation; **cost** and **time** show the total solution cost reported by META4ICS and the time needed for its resolution in milliseconds; **solution** shows the set of critical nodes expressed as a list of nodes with their respective costs (weights). These experiments have performed on a MacBook Pro (15-inch, 2018), 2.9 GHz Intel Core i9, 32 GB 2400 MHz DDR4.

As a final remark, it can be observed that some cases with the same size and composition parameters have very different resolution times. This is an interesting phenomenon and it is due to the internal logical composition of the AND/OR graph and how well the underlying solver performs with each case. Within our experiments, we have observed that none of the two solvers used in META4ICS is faster than the other in all of the cases. We believe this is an interesting problem that should be further investigated in the context of MaxSAT solvers.

id	gNodes	gAT	gAND	gOR	tsVars	tsClauses	cost	time	solution
1	5000	3977	512	512	8978	23981	2	1163	[(14:2)]
2	5000	3967	529	505	8968	23971	16	1264	[(2:16)]
3	5000	3984	515	502	8985	23988	2	920	[(98:2)]
4	5000	4008	461	532	9009	24012	8	919	[(8:1),(961:7)]
5	5000	3990	510	501	8991	23994	1	930	[(4057:1)]
6	5000	4002	486	513	9003	24006	5	946	[(788:3),(8431:2)]
7	5000	4026	477	498	9027	24030	20	920	[(8:1),(759:12),(6706:3),(6860:1),(7304:3)]
8	5000	3996	518	487	8997	24000	6	894	[(4:6)]
9	5000	4046	488	467	9047	24050	5	904	[(3177:5)]
10	5000	4026	493	482	9027	24030	5	887	[(4029:4),(5503:1)]
11	5000	3029	979	993	8030	23033	31	880	[(1258:30),(2189:1)]
12	5000	2998	1012	991	7999	23002	68	935	[(920:2),(1128:3),(1324:9),(1926:3),(2690:13),(2773:35),(3441:3)]
13	5000	3021	1008	972	8022	23025	2	879	[(300:2)]
14	5000	2986	1020	995	7987	22990	46	956	[(1027:12),(3624:3),(3778:17),(6453:2),(6496:4),(6544:8)]
15	5000	3014	1006	981	8015	23018	152	1889	[(5:1),(2930:5),(3245:7),(3434:34),(3925:17),(4158:14), (4457:34), (6307:19),(7048:5),(7177:2),(7186:1), (7191:7),(7337:1),(7367:1),(7570:4)]
16	5000	3030	990	981	8031	23034	12	859	[(3:12)]
17	5000	3015	1034	952	8016	23019	3	867	[(4:2),(7561:1)]
18	5000	3008	956	1037	8009	23012	22	877	[(2:19),(6253:3)]
19	5000	3031	970	1000	8032	23035	8	865	[(5444:8)]
20	5000	3033	972	996	8034	23037	13	870	[(2:13)]
21	10000	7983	1017	1001	17984	47987	1	1126	[(16139:1)]
22	10000	8026	996	979	18027	48030	13	1313	[(2:13)]
23	10000	8024	991	986	18025	48028	12	1310	[(24:1),(1580:1),(2201:2),(9663:3),(14360:5)]
24	10000	8058	974	969	18059	48062	14	1184	[(5:9),(12130:2),(12207:2),(12422:1)]
25	10000	8026	989	986	18027	48030	7	1459	[(9009:3),(12030:4)]
26	10000	7984	1003	1014	17985	47988	12	1309	[(7:9),(17023:1),(17074:2)]
27	10000	8051	939	1011	18052	48055	50	6248	[(1997:4),(2744:9),(3398:17),(5610:6),(6304:1),(11315:1),(11771:4), (14136:1),(16399:4),(16831:3)]
28	10000	7998	995	1008	17999	48002	3	1141	[(2:3)]
29	10000	8095	972	934	18096	48099	10	1224	[(8330:4),(11696:5),(13881:1)]
30	10000	8022	987	992	18023	48026	2	1178	[(2:2)]
31	10000	6013	1991	1997	16014	46017	49	1275	[(4:32),(4857:17)]
32	10000	5981	2039	1981	15982	45985	7	1098	[(5929:5),(6199:2)]
33	10000	6023	1957	2021	16024	46027	12	1127	[(3:12)]
34	10000	6015	2021	1965	16016	46019	60	1247	[(5641:17),(5858:3),(5969:16),(5997:6),(6025:6),(6033:3), (6133:2),(9790:2),(11731:5)]
35	10000	5953	1975	2073	15954	45957	24	1071	[(2:24)]
36	10000	6030	2034	1937	16031	46034	36	1279	[(12930:22),(15804:2),(15947:12)]
37	10000	6008	1996	1997	16009	46012	3	1092	[(13780:3)]
38	10000	6054	1963	1984	16055	46058	1	1111	[(3447:1)]
39	10000	6015	1977	2009	16016	46019	27	1194	[(2391:27)]
40	10000	6042	1967	1992	16043	46046	73	2607	[(2820:1),(7720:3),(8418:14),(8586:2),(10093:46),(12532:7)]

TABLE II
BENCHMARK DESCRIPTION - CASES 1 TO 40

id	gNodes	gAT	gAND	gOR	tsVars	tsClauses	cost	time	solution
41	15000	12090	1443	1468	27091	72094	16	3024	[(97:8),(6443:8)]
42	15000	11973	1517	1511	26974	71977	3	2139	[(14702:3)]
43	15000	12004	1529	1468	27005	72008	2	1511	[(1021:1),(22132:1)]
44	15000	11969	1517	1515	26970	71973	18	10965	[(2:18)]
45	15000	12123	1457	1421	27124	72127	5	1576	[(2350:1),(2665:1),(3626:3)]
46	15000	11969	1480	1552	26970	71973	5	1564	[(13:4),(25198:1)]
47	15000	11949	1490	1562	26950	71953	9	1605	[(58:1),(3272:6),(6793:2)]
48	15000	11982	1542	1477	26983	71986	18	2863	[(5:18)]
49	15000	12015	1486	1500	27016	72019	3	1511	[(2649:1),(6731:2)]
50	15000	11980	1496	1525	26981	71984	1	1627	[(1853:1)]
51	15000	9043	2893	3065	24044	69047	11	1423	[(22630:2),(22728:9)]
52	15000	9036	2976	2989	24037	69040	123	9571	[(10932:44),(13135:5),(15118:5),(15681:3),(15695:66)]
53	15000	9046	2938	3017	24047	69050	14	1182	[(3:6),(14013:8)]
54	15000	9015	3028	2958	24016	69019	47	1502	[(3:42),(11757:2),(13221:3)]
55	15000	8987	3035	2979	23988	68991	8	1208	[(17563:3),(17567:5)]
56	15000	9015	3002	2984	24016	69019	1	1483	[(15321:1)]
57	15000	9099	2966	2936	24100	69103	80	5040	[(3352:8),(3770:4),(8682:5),(10152:8),(10159:3),(10239:1), (10468:3), (18507:6),(18555:8),(18573:2),(18628:8),(18753:14), (19274:5),(22843:2),(23171:3)]
58	15000	9011	2943	3047	24012	69015	4	1152	[(18688:2),(19011:2)]
59	15000	9035	2998	2968	24036	69039	53	10802	[(2:53)]
60	15000	9013	3066	2922	24014	69017	14	1264	[(11895:1),(12533:3),(12590:3),(16787:2),(23583:5)]
61	20000	16004	1975	2022	36005	96008	1	1637	[(16475:1)]
62	20000	15977	1958	2066	35978	95981	3	1686	[(34706:3)]
63	20000	16007	2071	1923	36008	96011	1	2040	[(32144:1)]
64	20000	16067	1978	1956	36068	96071	1	1572	[(35189:1)]
65	20000	15999	1990	2012	36000	96003	2	1873	[(12825:2)]
66	20000	15947	2037	2017	35948	95951	4	2074	[(3:4)]
67	20000	16019	1992	1990	36020	96023	7	2153	[(6:3),(20228:4)]
68	20000	15976	2036	1989	35977	95980	5	1492	[(7:2),(17657:2),(24220:1)]
69	20000	16019	1997	1985	36020	96023	9	2545	[(3:9)]
70	20000	16052	1909	2040	36053	96056	2	2031	[(29145:2)]
71	20000	12037	4033	3931	32038	92041	2	1847	[(21278:2)]
72	20000	11977	3997	4027	31978	91981	373	12887	[(1172:17),(11919:5),(12728:2),(1295:1),(13840:2),(14081:4), (14453:22), (15461:5),(16291:13),(17369:3),(18459:20),(19920:1), (19925:4),(20651:2), (20789:37),(20929:9),(21080:41),(22175:7), (22642:1),(22804:27),(22806:12),(22809:3),(22818:26),(22829:6), (22852:31),(23065:7),(28543:65)]
73	20000	12037	3999	3965	32038	92041	3	1513	[(22359:2),(30489:1)]
74	20000	12004	3980	4017	32005	92008	1	1434	[(2:1)]
75	20000	12059	3930	4012	32060	92063	28	3071	[(2:18),(26845:10)]
76	20000	12094	3858	4049	32095	92098	2	1772	[(6431:2)]
77	20000	12000	4014	3987	32001	92004	3	1474	[(27418:3)]
78	20000	12110	3940	3951	32111	92114	4	1665	[(4:4)]
79	20000	12036	4012	3953	32037	92040	71	1600	[(2:9),(14499:3),(17840:14),(19998:32),(28910:2), (29045:9),(29937:2)]
80	20000	12035	4055	3911	32036	92039	8	1688	[(13697:8)]

TABLE III
BENCHMARK DESCRIPTION - CASES 41 TO 80

REFERENCES

[1] M. Barrère, C. Hankin, N. Nicolaou, D. Eliades, and T. Parisini, "Identifying Security-Critical Cyber-Physical Components in Industrial Control Systems," https://arxiv.org/abs/1905.04796, May 2019.

[2] Y. Desmedt and Y. Wang, "Maximum Flows and Critical Vertices in AND/OR Graphs," in *Computing and Combinatorics*, O. H. Ibarra and L. Zhang, Eds. Springer Berlin Heidelberg, 2002, pp. 238–248.

[3] ——, "Analyzing Vulnerabilities Of Critical Infrastructures Using Flows And Critical Vertices In And/Or Graphs," *Int. J. Found. Comput. Sci.*, vol. 15, no. 1, pp. 107–125, 2004.

[4] G. Jakimoski and M. Burmester, "Using Faulty Flows in AND/OR Graphs to Model Survivability and Reliability in Distr. Systems," 2004.

[5] U. dos Santos Souza, F. Protti, and M. D. da Silva, "Revisiting the complexity of and/or graph solution," *Journal of Computer and System Sciences*, vol. 79, no. 7, pp. 1156 – 1163, 2013.

[6] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, ser. RAND Corporation research study. University Press, 1962.

[7] G. S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, A. Slisenko, Ed., 1970, pp. 234–259.

[8] J. Davies and F. Bacchus, "Solving MAXSAT by Solving a Sequence of Simpler SAT Instances," in *Principles and Practice of Constraint Programming – CP 2011*, J. Lee, Ed. Springer, 2011, pp. 225–239.

[9] M. Barrère, "META4ICS - Metric Analyser for Industrial Control Systems," https://github.com/mbarrere/meta4ics, May 2019.

[10] "SAT4J," http://www.sat4j.org/, Cited June 2019.

MaxSAT Queries in The Design of Interpretable Rule-based Classifiers

Bishwamittra Ghosh
School of Computing
National University of Singapore
Singapore
bishwa@comp.nus.edu.sg

Dmitry Malioutov
T.J. Watson IBM Research center
dmal@alum.mit.edu

Kuldeep S. Meel
School of Computing
National University of Singapore
Singapore
meel@comp.nus.edu.sg

In machine learning, interpretability has gained significant attention in recent years. Rule-based classifiers are particularly effective for providing interpretations to end users. We have designed a rule-based classifier MLIC [1] and its incremental version IMLI [2] to generate classification rules in CNF/DNF, wherein we formulate the learning problem as an optimization problem, specifically as a MaxSAT problem.

In our experiments, we use several benchmarks/datasets from UCI and Kaggle repository. We consider three parameters in our formulation: data-fidelity parameter $\lambda \in \{1, 5, 10, 15, 20\}$, number of clauses $k \in \{1, 2, 3, 4, 5\}$ in the desired rule, and rule-type $\in \{\text{CNF}, \text{DNF}\}$. We use 10-fold (fold-index $\in \{0, \ldots, 9\}$) cross-validation for evaluating the performance of our classifier with other state of the art classifiers. Additionally, we compute the accuracy of our classifier by constructing a MaxSAT instance and eventually solving it. Therefore, each MaxSAT instance has the following naming convention.

$\langle$dataset$\rangle$_$\langle\{$train, test$\}\rangle$_$\langle$fold$\rangle$_$\langle$rule-type$\rangle$_$\langle k\rangle$_$\langle\lambda\rangle$.wcnf

Construction of MaxSAT query: We are given a dataset $(\mathbf{X}, \mathbf{y})$ of n samples, each sample with m binary features ($\mathbf{X} \in \{0,1\}^{n\times m}$ and $y \in \{0,1\}^n$). We learn a k-clause CNF rule and consider $k \times m + n$ boolean variables $\{b^1_1, b^1_2, \ldots, b^1_m, \ldots, b^k_m, \eta_1, \ldots, \eta_n\}$. The MaxSAT query comprises of two different soft clauses: E_q and S^i_j and hard clause H_q. The weight of a clause is defined by $\mathsf{wt}(\cdot)$.

$$E_q := \neg\eta_i; \qquad \mathsf{wt}(E_q) = \lambda$$
$$S^i_j := \neg b^i_j; \qquad \mathsf{wt}(S^i_j) = 1$$
$$H_q := \neg\eta_q \rightarrow \Big(y_q \leftrightarrow \bigwedge_{i=1}^{k} \mathbf{X}_q \circ \mathbf{B}_i\Big); \qquad \mathsf{wt}(H_q) = \infty$$

Here λ is the data-fidelity parameter that handles the trade-off between rule-sparsity and prediction accuracy. In the hard clause H_q, $\mathbf{X}_q$ is the q-th row of input matrix $\mathbf{X}$, y_q is the q-th element of $\mathbf{y}$, and $\mathbf{B}_i = \{b^i_j \mid j \in \{1, \ldots, m\}\}$. Between two vectors $\mathbf{u}$ and $\mathbf{v}$ over boolean variables or constants (for example, $0, 1$), $\mathbf{u} \circ \mathbf{v}$ represents the inner product of $\mathbf{u}$ and $\mathbf{v}$, i.e. $\mathbf{u} \circ \mathbf{v} = \bigvee_i u_i \wedge v_i$, where u_i and v_i denote a variable/constant at the i-th index of $\mathbf{u}$ and $\mathbf{v}$ respectively.

Once we construct all soft and hard clauses, the MaxSAT query Q is the conjunction of all clauses.

$$Q := \bigwedge_{q=1}^{n} E_q \wedge \bigwedge_{i=1,j=1}^{i=k,j=m} S^i_j \wedge \bigwedge_{q=1}^{n} H_q$$

References

[1] D. Malioutov and K. S. Meel, "Mlic: A maxsat-based framework for learning interpretable classification rules," in *Proc. of CP*, 2018.

[2] B. Ghosh and K. S. Meel, "Imli: An incremental framework for maxsat-based learning of interpretable classification rules," in *Proc. of AIES*, 2019.

Datasets of Networks for Benchmarking MaxSAT Evaluation 2019

Said Jabbour
CRIL - CNRS UMR 8188
Université d'Artois
France
jabbour@cril.fr

Nizar Mhadhbi
CRIL - CNRS UMR 8188
Université d'Artois
France
mhadhbi@cril.fr

Badran Raddaoui
SAMOVAR, CNRS, Télécom SudParis
Institut Polythechnique de Paris
France
badran.raddaoui@telecom-sudparis.eu

Lakhdar Sais
CRIL - CNRS UMR 8188
Université d'Artois
France
sais@cril.fr

The analysis of large networks has become very usefull in a wide range of applications including social sciences, biology and complex systems. This paper describes a set of instances of networks for benchmarking MaxSAT Evaluation 2019. These datasets were used in [1], [2]. All instances provided here are wcnf formulae encoded in DIMACS wcnf format.

Amazon:
This instance represents a network that was collected by crawling Amazon website. It is based on Customers Who Bought This Item Also Bought feature of the Amazon website. If a product i is frequently co-purchased with product j, the graph contains an edge from i to j.

DBLP:
The DBLP computer science bibliography provides a comprehensive list of research papers in computer science. This instance represents a co-authorship network where two authors are connected if they publish at least one paper together.

Youtube:
Youtube is a video-sharing web site that includes a social network. This instance represents the Youtube social network where users form friendship each other and users can create groups which other users can join.

Railway:
This instance represents the Indian Railway network that consists of nodes representing stations, where two stations a and b are connected by an edge if there exists at least one train-route such that both a and b are scheduled halts on that route.

Football:
This instance represents the network of American football games between Division IA colleges during regular season Fall of 2000. The vertices in the graph represent teams (identified by their college names), and edges represent regular-season games between the two teams they connect.

Karate:
This instance represents the karate social network where the data was collected from the members of a university karate club by Wayne Zachary in 1977. Each node represents a member of the club, and each edge represents a tie between two members of the club.

RiskMap:
This instance represents a graph which is a map of the popular strategy board game,Risk. It is a political map of the Earth, divided into 42 territories, which are grouped into 6 continents. Therefore, the graph is comprised of 42 vertices and 83 edges.

Politics Book:
This instances describes a network where nodes represent books about USpolitics sold by the online bookseller Amazon.com while edges represent frequent co-purchasing of books by the samebuyers on Amazon.

References

[1] S. Jabbour, N. Mahdhbi, B. Raddaoui, and L. Sais, "Triangle-Driven Community Detection in Large Graphs Using Propositional Satisfiability," 32nd IEEE International Conference on Advanced Information Networking and Applications. pp. 437–444, 2018.

[2] S. Jabbour, N. Mahdhbi, B. Raddaoui, and L. Sais, "Detecting Highly Overlapping Community Structure by Model-based Maximal Clique Expansion," IEEE International Conference on Big Data. pp. 1031–1036, 2018.

Parametric RBAC Maintenance via Max-SAT: Benchmarks Description

Marco Mori
Bank of Italy
ICT Department, Centro Donato Menichella
Rome, Italy
Email: marco.mori@bancaditalia.it

Marco Benedetti
Bank of Italy
ICT Department, Centro Donato Menichella
Rome, Italy
Email: marco.benedetti@bancaditalia.it

***Abstract*—Many organizations have adopted a Role-Based Access Control model (RBAC) to reduce administration costs through an efficient management of permissions and the enforcement of basic security principles.**

In this approach, it is often needed to revise the current RBAC policies to incorporate new permission-to-user assignments that were not granted by mistake, or were not correctly anticipated.

This note describes a family of benchmarks which formalize such RBAC maintenance instances as Weighted Partial Max-SAT (WPMS) problems. A brief description is provided along with a definition of the problem domain, and details on the parameters used for generating the instances.

I. Benchmarks Description

Role-based Access Control (RBAC) [1], [2] is a widely adopted access control model which supports an efficient management of permissions by means of roles. Users are assigned a set of roles, each encapsulating the permissions required to accomplish certain tasks. It follows that an RBAC policy can be formalized by means of two Boolean matrices: the role-to-user assignment matrix and the permission-to-role assignment matrix.

In this context, we considered the RBAC Maintenance problem, i.e., the problem of fixing the current RBAC policy by incorporating a new permission-to-user assignment given as input (i.e., an "exception") which has not been already granted. In this process we balance two possibly conflicting metrics, i.e., the "simplicity" of the target policy and its "stability" with respect to the previous policy.

We formalize the role maintenance problem as a Weighted Partial (WPMS) Max-Sat instance [3] and we generate a set of benchmarks to prove the viability of our formalization in real-world scenarios. The maintenance benchmarks have been generated from four different matrices of permission-to-user assignments of increasing size. The smallest one is a tiny matrix representing a simple organizations (*smallcomp*); the remaining three matrices belong to three different datasets available in the role mining literature [4], namely *domino*, *university* and *firewall1*. Starting from each of the four matrices, we randomly subtract a fixed number of assignments and then synthesize an initial RBAC policy through Fastminer [5]. Thus, for each matrix, we obtain: (i) an RBAC policy "currently in-place", and (ii) a set of exceptions to incorporate (corresponding to the randomly removed assignments). From the previous inputs and taking into account several different values for the balancing parameter β (where β close to 0 means "try to remain as close as possible to the initial state" and β close to 1 means "try to optimize as much as possible the resulting state"), we generate the WPMS instances.

As far as the naming convention is concerned, each instance file is named by concatenating the strings "`role`" with the name of the dataset it refers to (either "`smallcomp`" or "`domino`" or "`university`" or "`firewall1`") with the value of the balancing parameter β with the index of the exception to incorporate. For example, the file named:

"`role_university_0.8_9.cnf`"

contains a WPMS instance in DIMACS format that encodes the problem of incorporating exception number 9 into the "`university`" benchmark using *0.8* as balancing factor.

II. Acknowledgements

We thank the original creators of the role-mining datasets and Ian M. Molloy for providing us with the corresponding set of standardized *domino*, *university* and *firewall1* matrices.

References

[1] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[2] B. Mitra, S. Sural, J. Vaidya, and V. Atluri, "A Survey of Role Mining," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 50, 2016.

[3] M. Benedetti and M. Mori, "Parametric RBAC Maintenance trough Max-SAT," in *Proceedings of the 23th ACM Symposium on Access Control Models and Technologies*. ACM, 2018, pp. 15–25.

[4] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo, "Evaluating Role Mining Algorithms," in *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*. ACM, 2009, pp. 95–104.

[5] J. Vaidya, V. Atluri, and J. Warner, "Roleminer: Mining Roles using Subset Enumeration," in *Proceedings of the 13th ACM Conference on Computer and Communications security*. ACM, 2006, pp. 144–153.

Maximum Common Sub-Graph Extraction

Miguel Terra-Neves
OutSystems
Portugal
miguel.neves@outsystems.com

Miguel Ventura
OutSystems
Portugal
miguel.ventura@outsystems.com

I. Introduction

This benchmark set is motivated by the problem of pattern detection in software code, which is a particular instance of the general maximum common sub-graph extraction problem. Other applications of maximum common sub-graphs include matching of chemical structures [4], computation of graph edit distances [1], [2] and synthesis of malware signatures [3]. Given a finite set of graphs $\mathbb{G}$, the maximum common sub-graph problem consists of determining the largest sub-graph that occurs in all of the graphs in $\mathbb{G}$. Note that the notion of *largest* may differ between different applications. We start by formally defining the maximum common sub-graph extraction problem, followed by a description of its encoding into a partial MaxSAT formula. We conclude by explaining how the benchmarks were generated and the file naming convention.

II. Problem Statement

A graph is given by a pair $G = (V, E)$, where V is its set of nodes and E its set of edges. We assume that G is a directed graph, i.e., if $(u, v) \in E$, then G contains an edge from u to v, but not necessarily from v to u (the latter is the case only if $(v, u) \in E$ as well). We consider that nodes and edges have types. Given a node $v \in V$, we denote v's type as $t(v)$. Analogously, we denote the type of an edge $(u, v) \in E$ as $t(u, v)$.

Let $G = (V, E)$ and $G' = (V', E')$ be graphs. We say that G is a sub-graph of G' if, and only if, there exists a mapping $F : V' \to V$ such that:

- For all $v \in V$, there exists $v' \in V'$ such that $t(v) = t(v')$ and $F(v') = v$.
- For all $u', v' \in V'$, if $u' \neq v'$ and $F(u') \neq \perp$ (or $F(v') \neq \perp$) then $F(u') \neq F(v')$. Note $F(u') = \perp$ if u' is not mapped to any node in V, which is the case for at least one node if V' is larger than V.
- For all $(u, v) \in E$, there exists $u', v' \in V'$ such that $F(u') = u$, $F(v') = v$, $(u', v') \in E$ and $t(u, v) = t(u', v')$.

Let $\mathbb{G} = \{G_1, G_2, \ldots, G_n\}$ be a set of graphs and $G_0 = (V_0, E_0)$ another graph. We say that G_0 is a common sub-graph of $\mathbb{G}$ if G_0 is a sub-graph of all graphs in $\mathbb{G}$. Given $\mathbb{G}$, the goal is to find its largest common sub-graph. In this particular scenario, we consider the largest common sub-graph to be the one that maximizes $|E_0|$.

III. Partial MaxSAT Encoding

Consider the graph set $\mathbb{G} = \{G_1, G_2, \ldots, G_n\}$. In order to build the MaxSAT formula, a graph in $\mathbb{G}$ is picked to be the reference graph $G_r = (V_r, E_r)$. A common sub-graph G_0 is extracted by computing a mapping of the nodes of the graphs in $\mathbb{G}_m = \mathbb{G} \setminus \{G_r\}$ to nodes of V_r.

Three sets of Boolean variables are considered:

- A domain variable d_i is introduced for each $v_i \in V_r$. If $d_i = 1$, then $v_i \in V_0$, otherwise $d_i = 0$.
- A mapping variable $f_{i,k,j}$ indicates if the node v_i of some graph $G_k \in \mathbb{G}_m$ is mapped to node $v_j \in V_r$. If so, then $f_{i,k,j} = 1$, otherwise $f_{i,k,j} = 0$. Note that $f_{i,k,j} = 1$ only makes sense if v_i and v_j share the same type. Therefore, if $t(v_i) \neq t(v_j)$, we assume $f_{i,k,j} = 0$.
- A control-flow variable $c_{i,j}$ is introduced for each pair of nodes $v_i, v_j \in V_r$. If $c_{i,j} = 1$, then $(v_i, v_j) \in E_0$, otherwise $c_{i,j} = 0$. Note that $c_{i,j} = 1$ only makes sense if $(v_i, v_j) \in E_r$. If $(v_i, v_j) \notin E_r$, we assume $c_{i,j} = 0$.

The MaxSAT formula encodes the following hard constraints:

- For all $v_i \in V_r$ and $G_k \in \mathbb{G}_m$ ($G_k = (V_k, E_k)$), v_i is a node of the common sub-graph (i.e. $v_i \in V_0$) if, and only if, at least one node $v_j \in V_k$ is mapped to v_i:

$$\forall_{v_i \in V_r} \forall_{G_k \in \mathbb{G}_m} : \left(\overline{d_i} \vee \bigvee_{v_j \in V_k} f_{i,k,j} \right) \wedge \bigwedge_{v_j \in V_k} \left(d_i \vee \overline{f_{i,k,j}} \right). \quad (1)$$

- For all $G_k \in \mathbb{G}_m$ and $v_p, v_q \in V_k$ such that $v_p \neq v_q$, v_p and v_q cannot be mapped to the same node of V_r:

$$\forall_{G_k \in \mathbb{G}_m} \forall_{v_p \in V_k} \forall_{v_q \in V_k, v_p \neq v_q} \forall_{v_i \in V_r} : \left(\overline{f_{p,k,i}} \vee \overline{f_{q,k,i}} \right). \quad (2)$$

- For all $G_k \in \mathbb{G}_m$ and $v_p \in V_k$, v_p is mapped to at most one node in V_r:

$$\forall_{G_k \in \mathbb{G}_m} \forall_{v_p \in V_k} \forall_{v_i \in V_r} \forall_{v_j \in V_r, v_i \neq v_j} : \left(\overline{f_{p,k,i}} \vee \overline{f_{p,k,j}} \right). \quad (3)$$

- For all $(v_i, v_j) \in E_r$, $(v_i, v_j) \in E_0$ if, and only if, for all $G_k \in \mathbb{G}_m$, there exists $v_p, v_q \in V_k$ such that v_p and v_q are mapped to v_i and v_j respectively, $(v_p, v_q) \in E_k$ and $t(v_i, v_j) = t(v_p, v_q)$. Let $E_k(t(v_i, v_j))$ denote the set of

edges in E_k that share the type $t(v_i, v_j)$. This constraint is encoded by the following formula:

$$\forall_{(v_i,v_j)\in E_r} \forall_{G_k\in\mathbb{G}_m} \forall_{(v_p,v_q)\in V_k\times V_k\setminus E_k(t(v_i,v_j))} : \left(\overline{f_{p,k,i}} \vee \overline{f_{q,k,j}} \vee \overline{c_{i,j}}\right). \quad (4)$$

- For all $(v_i, v_j) \in E_r$, if $(v_i, v_j) \in E_0$, then $v_i \in V_0$ and $v_j \in V_0$:

$$\forall_{(v_i,v_j)\in E_r} : (\overline{c_{i,j}} \vee d_i) \wedge (\overline{c_{i,j}} \vee d_j). \quad (5)$$

- For all $v_i \in V_r$, $v_i \in V_0$ if, and only if, there exists an edge $(u, v) \in E_0$ such that $u = v_i$ or $v = v_i$:

$$\forall_{v_i\in V_r} : \left(\overline{d_i} \vee \bigvee_{(u,v_j)\in E_r, u=v_i} c_{i,j} \vee \bigvee_{(v_j,u)\in E_r, u=v_i} c_{j,i}\right). \quad (6)$$

Recall that the goal is to maximize the number of edges in E_0, and thus the soft clause set is given by:

$$\bigcup_{(v_i,v_j)\in E_r} \{(c_{i,j})\}. \quad (7)$$

IV. Benchmark Generation and File Name Convention

The maximum common sub-graph MaxSAT benchmarks were generated from instances of pattern detection in real-world software code. For each maximum common sub-graph instance, given by a set of graphs $\mathbb{G}$, the graph with the least amount of nodes was chosen to be the reference graph. The file name for instance $\mathbb{G}$ follows the following convention. It starts with the string $\mathtt{g}N$, where $N = |\mathbb{G}|$, followed by a sequence of strings, one per graph $G_k \in \mathbb{G}$, separated by underscores. Each such string has the form $\mathtt{n}N_1\mathtt{e}N_2$, where $N_1 = |V_k|$ and $N_2 = |E_k|$.

References

[1] Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters 18(8), 689–694 (1997)

[2] Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters 19(3-4), 255–259 (1998)

[3] Feng, Y., Bastani, O., Martins, R., Dillig, I., Anand, S.: Automated synthesis of semantic malware signatures using maximum satisfiability. In: Annual Network and Distributed System Security Symposium (2017)

[4] Raymond, J.W., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. Journal of Computer-Aided Molecular Design 16(7), 521–533 (2002)

Solver Index

Benchmark Index

Author Index